EOARD

SPC - 98 - 4061

Lattice Boltzmann Models for Multicomponent Fluids

Contract No. F61775 - 98 - WE101

item #003 - Final Report

Delivery date: July 20, 1999

Principal investigator

Dr. Victor SOFONEA

Polytechnical University of Timişoara

Romania

EOARD

SPC – 98 – 4061

Lattice Boltzmann Models for Multicomponent Fluids

Contract No. F61775 – 98 – WE101

item #003 - Final Report

Delivery date: July 20, 1999

Principal investigator

Dr. Victor SOFONEA

Polytechnical University of Timişoara

Romania

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	20 Jul 1999	Final Report

4. TITLE AND SUBTITLE

Lattice Boltzmann Models for Multicomponent Fluids

5. FUNDING NUMBERS

F61775-98-WE101

6. AUTHOR(S)

Dr. Victor Sofonea

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Politehnica University of Timisoara
Bd. Mihai Viteazul 1
Timisoara 1900
Romania

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0200

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

SPC 98-4061

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words)

This report results from a contract tasking Politehnica University of Timisoara as follows: The contractor will investigate the utility of Lattice Boltzmann methods in the modeling of complex (multicomponent) fluids. A two-dimensional Lattice Boltzmann model will be developed, the geometry of which is not related to the thermal velocities of fluid particles. Comprehensive testing and analysis of the model, as well as benchmarking against existing Lattice Boltzmann models will be accomplished. Initial testing will be accomplished by modeling one-component fluid flow in two spatial dimensions, followed by study of a two-component flow in two dimensions. Finally, interparticle interactions will be added to allow investigation of interfaces between multiphase fluids. Two specific physical problems investigated: the shape of a sessile drop on a horizontal surface subjected to a gravitational field, and the effect of surface tension on contact angle.

14. SUBJECT TERMS

EOARD, Fluid Dynamics, Lattice Boltzmann Methods, Computational Mathematics

15. NUMBER OF PAGES

226

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19, SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

Chapter 1

Finite Difference Schemes for the Boltzmann Equation

1.1 Introduction

In the intermediate report [1], Centered Space (CS) finite difference schemes were developed to solve the Lattice Boltzmann (LB) evolution equations for the distribution functions $f_i(\mathbf{r}, t)$ of an one component fluid system, when the number of non - vanishing discrete velocities was $N = 6$ (the so - called seven bit model) or $N = 8$ (the so - called nine bit model):

$$
\begin{aligned}
f_i(\mathbf{r}, t + \delta t) \;\simeq\; & f_i(\mathbf{r}, t) - \frac{\delta t}{\tau_p}\,[\, f_i(\mathbf{r}, t) - f_i^{eq}(\mathbf{r}, t)\,] - \\
& \delta t\,\mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r}, t) + \frac{\delta t}{k_B T}\,\mathbf{F}(\mathbf{r}, t) \cdot [\,\mathbf{e}_i - \mathbf{u}(\mathbf{r}, t)\,]\, f_i^{eq}(\mathbf{r}, t) \\
& i = 0, 1, \ldots N
\end{aligned}
\tag{1.1}
$$

In the above equation, as well as in the whole report to follow, there is no implicit summation over the index i. We will use further the implicit summation rule (i.e., summation over repeated indices) only for cartesian components denoted by Greek letters α, β, γ ..., while summation over other indices (including i) will be always explicited, when such operations are present.

The validity of the CS finite difference schemes was successfully tested for two kinds of $2D$ flows between parallel plates: Poisseuille flow and Couette flow. The correct velocity profiles in the stationary regime (the parabolic

one, for Poisseuille flow, and the linear one, for Couette flow), as well as the right viscosity value were recovered for $N = 6$, as well as for $N = 8$ [1].

Our early attempts to use the CS scheme to simulate the diffusion phenomenon in a two component system revealed an unstable behavior. When this scheme is used to get the time dependent solutions of the lattice Boltzmann equations (1.1), the probability functions $f_i(x,t)$ become negative very often, forcing the simulation code to be stopped. Although centered finite difference schemes for spatial derivatives arise in a natural way and thus, are easy to introduce, these schemes are known to be unconditionally unstable in accordance to the von Neumann stability analysis [2, 3, 4] applied to hyperbolic equations like the Lattice Boltzmann equations (1.1).

This chapter reports our attempts to find appropriate finite difference schemes for the Lattice Boltzmann equations. These schemes are introduced for the nine bit model using the characteristics curves of hyperbolic equations in one space dimension [4] and we show that all these schemes are equivalent to the former Lattice Gas Lattice Boltzmann (LGLB) scheme [1] when the Courant - Friedrichs - Lewy number CFL (to be defined further) equals unity. All these schemes are based on Lagrange interpolation procedures. Some considerations relative to the treatment of boundary lattice nodes are also introduced here.

1.2 Spatial derivatives on a square lattice

To solve Eqs (1.1) on a square lattice, we need a procedure to compute the following term at the point $\mathbf{r} = (x, y)$

$$\begin{aligned} \mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r},t) &= (\mathbf{e}_i)_\alpha \partial_\alpha f_i(\mathbf{r},t) \\ &= (\mathbf{e}_i)_x \partial_x f_i(x,y,t) + (\mathbf{e}_i)_y \partial_y f_i(x,y,t) \end{aligned} \tag{1.2}$$

(implicit summation over cartesian components expressed by Greek indices is always understood).

A possibility is to use the Centered Space (CS) finite difference scheme

$$\partial_x f_i(x,y,t) \simeq \frac{f_i(x+\delta x,y,t) - f_i(x-\delta x,y,t)}{2\delta x} \tag{1.3}$$

where δx is the lattice spacing in the x direction. A similar expression holds also for the derivative in the y direction (∂_y); the lattice spacings are all

equal ($\delta x = \delta y$) in the case of the square lattice. As mentioned before, this centered finite difference scheme was found to be inadequate for the simulation of diffusion phenomena in a two component system.

Another possibility is to use the First Order Upwind (FU) finite difference scheme [3]. This scheme takes into account the fact that information propagates along the vector $\mathbf{e}_i$ and thus, one should consider also the sign of its components when writing the corresponding numerical scheme. For the x component, we have

$$\partial_x f_i(x,y,t) \simeq \begin{cases} \dfrac{f_i(x,y,t) - f_i(x-\delta_x,y,t)}{\delta x} & , \quad (\mathbf{e}_i)_x > 0 \\[2ex] \dfrac{f_i(x+\delta x,y,t) - f_i(x,y,t)}{\delta x} & , \quad (\mathbf{e}_i)_x < 0 \end{cases} \tag{1.4}$$

A Second Order Upwind (SU) finite difference scheme may be also introduced [3]

$$\partial_x f_i(x,y,t) \simeq \begin{cases} \dfrac{-3f_i(x,y,t) + 4f_i(x+\delta_x,y,t) - f_i(x+2\delta_x,y,t)}{2\delta x} \\[1ex] (\mathbf{e}_i)_x > 0 \\[1ex] \dfrac{3f_i(x,y,t) - 4f_i(x-\delta_x,y,t) + f_i(x-2\delta_x,y,t)}{2\delta x} \\[1ex] (\mathbf{e}_i)_x < 0 \end{cases} \tag{1.5}$$

Although the FU scheme was found to be stable, it has the disadvantage of a lattice spacing dependence of the viscosity. This is a general feature of first order schemes (see Section 1.10). In the case of second order schemes (like CS and SU), no lattice spacing dependence of the viscosity is observed, but these schemes are not stable when dealing with sharp interfaces (see next chapter, where the use of these schemes to simulate a diffusion couple is discussed). Other schemes, which are mainly based on the characteristics curve and use an interpolation procedure, are described below.

1.3 Characteristics

We start from the differential form of the LB equations (1.1)

$$\partial_t f_i(\mathbf{r},t) \; + \; \mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r},t) \; = \; q_i(\mathbf{r},t)$$
$$i \; = \; 0,\, 1,\, \ldots \, N \tag{1.6}$$

where the source term is

$$\begin{aligned} q_i(\mathbf{r},t) \; &= \; -\frac{1}{\tau_p} \left[f_i(\mathbf{r},t) \; - \; f_i^{eq}(\mathbf{r},t) \right] + \\ & \frac{1}{k_B T} \, \mathbf{F}(\mathbf{r},t) \cdot \left[\mathbf{e}_i \; - \; \mathbf{u}(\mathbf{r},t) \right] \, f_i^{eq}(\mathbf{r},t) \\ & i \; = \; 0,\, 1,\, \ldots \, N \end{aligned} \tag{1.7}$$

The homogeneous part of Eqs. (1.6)

$$\frac{D}{Dt} \, f_i(\mathbf{r},t) \; = \; \partial_t f_i(\mathbf{r},t) \, + \, \mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r},t) \; = \; 0 \tag{1.8}$$

has the solution

$$f_i(\mathbf{r},t) \; = \; f_i(\mathbf{r_0},t_0) \; = \; \text{constant} \tag{1.9}$$

since the vectors $\mathbf{e}_i$ are all constant. Consequently, for each $i = 0, 1, \ldots N$, the current point moves across a straight line (the characteristics line) which passes through the node $\mathbf{r}$ of the lattice at the moment t and has the orientation of the vector $\mathbf{e}_i$. The formal solution of the inhomogeneous evolution equations (1.6) may be written as

$$\begin{aligned} f_i(\mathbf{r},t+\delta t) \; &= \; f_i(\mathbf{r} \; - \; \mathbf{e}_i \delta t, \, t) \; - \; \frac{1}{\tau_p} \int_t^{t+\delta t} \left[f_i(\mathbf{r}(t'),t') \; - \; f_i^{eq}(\mathbf{r}(t'),t') \right] dt' \\ & + \; \frac{1}{k_B T} \int_{\delta t}^{t+\delta t} \mathbf{F}(\mathbf{r}(t'),t) \cdot \left[\mathbf{e}_i \; - \; \mathbf{u}(\mathbf{r}(t'),t') \right] \, f_i^{eq}(\mathbf{r}(t'),t') \\ & i \; = \; 0,\, 1,\, \ldots \, N \end{aligned} \tag{1.10}$$

where the integrals are calculated along the characteristics lines. At the initial moment $t' = t$, the current point is $\mathbf{r}(t') = \mathbf{r} - \mathbf{e}_i \delta t$ and thus, the

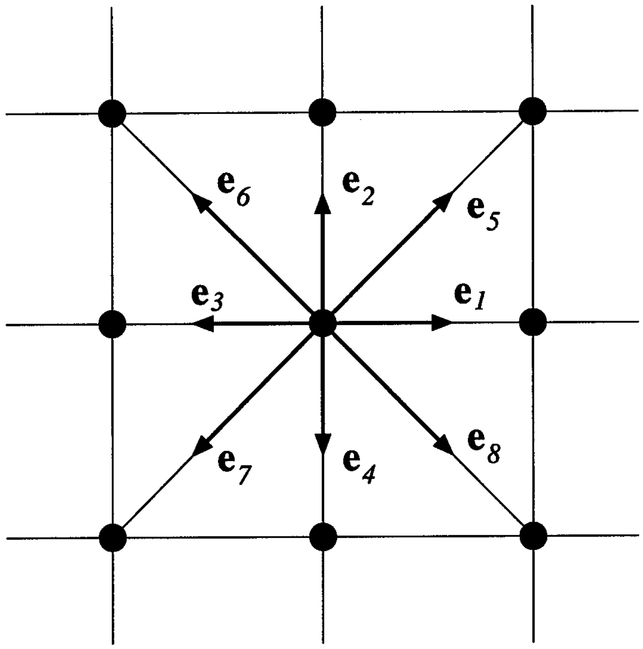


Figure 1.1: Velocity set in the nine bit ($N = 8$) LB model.

integrals in the equation above may be approximated using the rectangle formula to get

$$
\begin{aligned}
f_i(\mathbf{r}, t+\delta t) \;\simeq\;& f_i(\mathbf{r} - \mathbf{e}_i\delta t, t) - \frac{\delta t}{\tau_p}\left[f_i(\mathbf{r}-\mathbf{e}_i\delta t, t) - f_i^{eq}(\mathbf{r}-\mathbf{e}_i\delta t, t)\right] \\
&+ \frac{\delta t}{k_B T}\mathbf{F}(\mathbf{r} - \mathbf{e}_i\delta t, t)\cdot\left[\mathbf{e}_i - \mathbf{u}(\mathbf{r}-\mathbf{e}_i\delta t, t)\right] f_i^{eq}(\mathbf{r}-\mathbf{e}_i\delta t, t) \\
& i = 0, 1, \ldots N
\end{aligned}
\tag{1.11}
$$

This equation provides the key for the *characteristics – based* finite difference schemes to be developed further. Eqs. (1.11) are the expression of the well known Lagrange representation in classical fluid mechanics, where the evolution of a particle on its trajectory is described by the total time derivative.

When using the nine bit LGLB scheme on a square lattice, the characteristics lines always pass through the lattice nodes at any time step. These lines are associated to each velocity speed $\mathbf{e}_i$ in Figure 1.1. For further discussion,

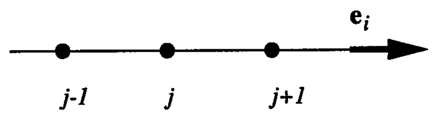


Figure 1.2: Lattice points along the characteristics line.

we will refer to one of these characteristics lines (which corresponds to any of the eight velocities $\mathbf{e}_i$ in Figure 1.1). We may choose a one dimensional reference system whose positive direction is orientated along the speed $\mathbf{e}_i$ and we represent the current points situated along this line as in figure 1.2. In this figure, the current point at the moment t is denoted by j, the next point on the characteristics line (which becomes the current point at the moment $t + \delta t$) is denoted $j + 1$, while the preceding one is denoted $j - 1$. We should remark the fact that the point j (which is the current point on the characterisctics line at the moment t) always refers to a lattice node, but the points $j - 1$ and $j + 1$ are superposed to lattice nodes only when using the LGLB scheme, as seen in figure 1.3. This figure shows the current point on the characteristics line, at different time steps, in the $X - t$ plane, when the LGLB scheme is used.

When δx is the spacing between the latice nodess on the reference axis X of the characteristics line (note that δx may be equal to l or to $l\sqrt{2}$, where

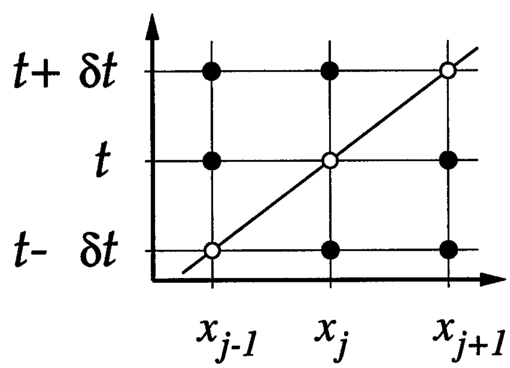


Figure 1.3: Current points (∘) on the characteristics line, in the LGLB model.

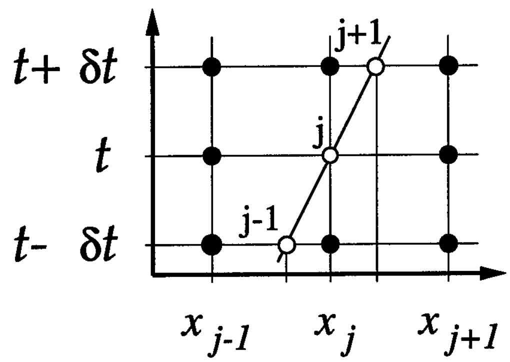


Figure 1.4: Current points (∘) on the characteristics line, in the Finite Difference LB model (CFL < 1).

l is the side length of the elementary square lattice cell) we have

$$\delta x = x_j - x_{j-1} = x_{j+1} - x_j \tag{1.12}$$

In the LGLB model, where particles are considered to propagate between lattice nodes during one time step, we have

$$\delta x = c\delta t \tag{1.13}$$

where c is the magnitude of the propagation speed $\mathbf{c}$ ($\mathbf{c} \in \{\mathbf{e}_i\}$, $i = 1, 2 \ldots N$) and δt is the time step.

When the lattice spacing δx does no more equal the product $c\delta t$, the current nodes $j-1$ and $j+1$ on the characteristics line passing through the lattice node x_j (i.e., node j) are no more superposed to lattice nodes. These current points lie now between the lattice node and the neighbors x_{j-1} or x_{j+1}, respectively. This general case is shown in figure 1.4 and is characterized by the Courant - Friedrichs - Lewy number

$$\mathsf{CFL} = \frac{c\delta t}{\delta x} \tag{1.14}$$

When CFL $= 1$, the former LGLB case is recovered. In accordance to the general theory of hyperbolic equations [2, 3, 4], a necessary (but not sufficient) condition to be satisfied by the finite difference scheme to be stable is

$$\mathsf{CFL} \leq 1 \tag{1.15}$$

1.4 Interpolation procedures on the characteristics line

The Lattice Boltzmann equations (1.11) may be solved using an iterative procedure. The value $f_i(\mathbf{r} - \mathbf{e}_i\delta t, t)$ of the distribution function (as well as the value $\mathbf{u}(\mathbf{r} - \mathbf{e}_i\delta t, t)$ of the local velocity) may be calculated using an interpolation procedure along the characteristics line which passes through the lattice node $\mathbf{r}$ at time $t + \delta t$. The current point at the moment t is $\mathbf{r} - \mathbf{e}_i\delta t$, which corresponds to the point x in figure 1.5. If we know the values of the distribution function at at the moment t in two lattice nodes x_1 and x_2, we may use the first order Lagrange interpolation formula to evaluate the distribution function in the point x

$$f_i(x,t) = \frac{x - x_2}{x_1 - x_2} f_i(x_1,t) + \frac{x - x_1}{x_2 - x_1} f_i(x_2,t) \tag{1.16}$$

The second order Lagrange interpolation formula may be used when three values of the distribution functions are known

$$\begin{aligned} f_i(x,t) &= \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f_i(x_1,t) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f_i(x_2,t) \\ &+ \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f_i(x_3,t) \end{aligned} \tag{1.17}$$

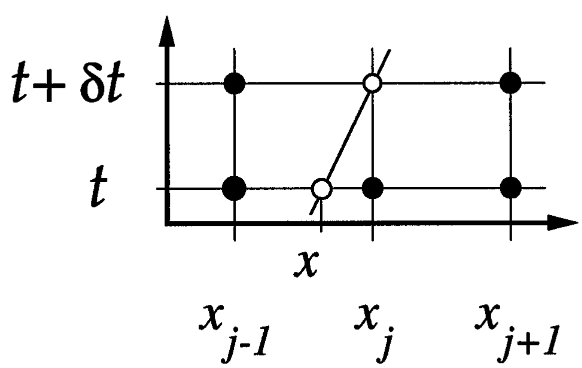


Figure 1.5: Current points (∘) on the characteristics line.

Following finite difference schemes may be introduced, depending on the choice of the interpolation formula, as well as on the number and the location of the interpolation points:

1. *Upwind scheme:* the first order interpolation formula (1.16) is used with $x_1 \equiv x_{j-1}$ and $x_2 \equiv x_j$.

2. *Lax Friedrichs scheme:* the first order interpolation formula (1.16) is used with $x_1 \equiv x_{j-1}$ and $x_2 \equiv x_{j+1}$.

3. *Lax Wendroff scheme:* the second order interpolation formula (1.17) is used with $x_1 \equiv x_{j-1}$, $x_2 \equiv x_j$ and $x_3 \equiv x_{j+1}$.

These finite difference schemes, as well as the *space centered scheme* are discussed below.

1.5 Upwind scheme

The upwind scheme is recovered when using the first order interpolation formula (1.16) with

$$\begin{aligned} x &= x_j - c\delta t \\ x_1 &= x_{j-1} \\ x_2 &= x_j \end{aligned} \tag{1.18}$$

In this case, the interpolation formula (1.16) gives

$$f_i(x_j - c\delta t, t) = f_i(x_j, t) - \mathsf{CFL}\,[\,f_i(x_j, t) - f_i(x_{j-1}, t)\,] \tag{1.19}$$

A particular attention should be given to boundary nodes. When the vector $\mathbf{e}_i$ is pointing outwards the lattice domain, the interpolation formula (1.19) may be used. But the interpolation formula should be somewhat different when th vector $\mathbf{e}_i$ points into the lattice domain: if x_j is a node located on the boundary, there is no corresponding x_{j-1} node in this case. The node x_{j+1} may be used instead, and we have the following extrapolation formula in this case:

$$f_i(x_j - c\delta t, t) = f_i(x_j, t) - \mathsf{CFL}\,[\,f_i(x_{j+1}, t) - f_i(x_j, t)\,] \tag{1.20}$$

After interpolating all distribution functions f_i, $i = 0, 1 \ldots N$ in the current point $\mathbf{r} - \mathbf{e}_i\delta t$, the local velocity $\mathbf{u}(\mathbf{r} - \mathbf{e}_i\delta t, t)$ should be evaluated and introduced in Eq. (1.11) which allows to calculate the new values of the distribution functions in all lattice nodes.

Since the values of the velocity components are prescribed on the boundaries, an interpolation procedure becomes necessary to evaluate also the velocity components at the current point $\mathbf{r} - \mathbf{e}_i\delta t$ when the node $\mathbf{r}$ is located on the boundary. Interpolation and extrapolation formulae similar to Eqs. (1.19) and (1.20) give the value of the velocity $\mathbf{u}(x_j - c\delta t, t)$, which allows the calculation of the equilibrium distribution function $f_i^{eq}(x_j - c\delta t, t)$ in Eq. (1.11).

A very important feature of the upwind finite difference scheme (1.19) is the recovery of the former Lattice Gas like Latice Boltzmann (LGLB) model when $\mathsf{CFL} = 1$. The upwind scheme says that the information (i.e., the distribution function f_i) available at the moment t at the current point $x_j - c\delta t$ is processed there as a result of collisions (through the relaxation term mainly characterized by the physical relaxation time τ_p, as well as the action of some force F), and thereafter arrives in node x_j at the moment $t + \delta t$. But, when $\mathsf{CFL} = 1$, the initial current point $x_j - c\delta t$ is identical to the lattice node x_{j-1}, in accordance to Eq. (1.19). This feature belongs also to the Lax - Friedrichs and the Lax - Wendroff schemes to be discussed below. In this respect, all these *characteristics curve* derived finite difference schemes provide a natural generalisation of the former LGLB model and allow to separate the lattice spacing δx from the particles free path $c\delta t$. This separation process provides the keystone to the LB simulation of multicomponent particle systems.

1.6 Lax - Friedrichs scheme

The Lax - Friedrichs scheme [2] is recovered when using the first order interpolation formula (1.16) with the following interpolation points

$$\begin{aligned} x &= x_j - c\delta t \\ x_1 &= x_{j-1} \\ x_2 &= x_{j+1} \end{aligned} \tag{1.21}$$

The interpolation formula (1.16) gives, at the moment t

$$f_i(x_j - c\delta t, t) = \frac{f_i(x_{j+1}, t) + f_i(x_{j-1}, t)}{2}$$

$$- \quad \mathsf{CFL}\, \frac{f_i(x_{j+1},t) \, - \, f_i(x_{j-1},t)}{2} \tag{1.22}$$

At the boundary nodes, the Lax - Friedrichs scheme cannot be used, but the upwind scheme may be accepted instead.

1.7 Space centered scheme

If we approximate

$$\frac{f_i(x_{j+1},t) \, + \, f_i(x_{j-1},t)}{2} \approx f_i(x_j,t) \tag{1.23}$$

in the Lax Friedrichs scheme (1.22), we get the usual space centered scheme

$$f_i(x_j \, - \, c\delta t, t) \, = \, f_i(x_j,t) \, - \, \mathsf{CFL}\, \frac{f_i(x_{j+1},t) \, - \, f_i(x_{j-1},t)}{2} \tag{1.24}$$

This scheme is known the be unconditionally unstable [2, 3]. We really get negative values of the distribution functions when using this scheme, after a certain number of time steps, and this situation forces the LB code to be stopped. The number of time steps before the space centered computer code crashes is dependent on the CFL number, being larger when this number becomes smaller. Using a smaller Courant - Friedrichs - Lewy number for a given lattice spacing δx is unconvenient since this means a smaller time step δt, which generates a huge increase of the number of necessary time steps to be performed in order to recover the system evolution during a certain time interval.

1.8 Lax - Wendroff scheme

The second order Lagrange interpolation formula (1.17) and the following interpolation points

$$\begin{aligned} x &= x_j \, - \, c\delta t \\ x_1 &= x_{j-1} \\ x_2 &= x_j \\ x_3 &= x_{j+1} \end{aligned} \tag{1.25}$$

give the Lax - Wendroff finite difference scheme

$$\begin{aligned} f_i(x_j - c\delta t, t) &= \frac{\mathsf{CFL}\,(1 + \mathsf{CFL})}{2}\, f(x_{j-1}, t) \\ &- (\mathsf{CFL} - 1)\,(\mathsf{CFL} + 1)\, f_i(x_j, t) \\ &- \frac{\mathsf{CFL}\,(1 - \mathsf{CFL})}{2}\, f(x_{j+1}, t) \end{aligned} \tag{1.26}$$

As for the Lax - Friedrichs scheme, the Lax - Wendroff scheme (1.26) cannot be applied in the boundary nodes. One possibility is to use there the upwind schemes (1.19) and (1.20) for outgoing and ingoing speed vectors $\mathbf{e}_i$, respectively. Another possibility is to preserve the second order interpolation degree and to use the followig expressions

$$\begin{aligned} f_i(x_j - c\delta t, t) &= \frac{\mathsf{CFL}\,(\mathsf{CFL} - 1)}{2}\, f_i(x_{j-2}, t) \\ &+ \mathsf{CFL}\,(2 - \mathsf{CFL})\, f_i(x_{j-1}, t) \\ &+ \frac{(1 - \mathsf{CFL})\,(2 - \mathsf{CFL})}{2}\, f_i(x_j, t) \end{aligned} \tag{1.27}$$

$$\begin{aligned} f_i(x_j - c\delta t, t) &= \frac{(1 + \mathsf{CFL})\,(2 + \mathsf{CFL})}{2}\, f_i(x_j, t) \\ &- \mathsf{CFL}\,(2 + \mathsf{CFL})\, f_i(x_{j+1}, t) \\ &- \frac{\mathsf{CFL}\,(1 + \mathsf{CFL})}{2}\, f_i(x_{j+2}, t) \end{aligned} \tag{1.28}$$

for the outgoing and ingoing speed vectors $\mathbf{e}_i$, respectively. Note that, in these expressions, the node x_j is always on the lattice border, while the positive direction (from node x_j to node x_{j+1} is the same as the direction of the speed vector $\mathbf{e}_i$.

1.9 Interpolation Supplemented Schemes

1.9.1 General Description

One can see that the procedure described by Eq. (1.11) implies the following computation steps:

1. Lagrange interpolation to find the distribution functions, as well as the local velocities in the points $\mathbf{r} - \mathbf{e}_i\delta t$; these points are not lattice nodes and are all different for $i = 1, 2, \ldots N$

2. collisions processing and the addition of the effect of external forces

3. propagation from the point $\mathbf{r} - \mathbf{e}_i\delta t$ to the lattice node $\mathbf{r}$

For each lattice node $\mathbf{r}$, one has to perform $N \times (N+1)$ interpolations since there are N directions $\mathbf{e}_i$ and $N + 1$ distribution functions to be interpolated. The function f_0 should also be interpolated since its value in the point $\mathbf{r} - \mathbf{e}_i\delta t$ is needed to calculate the value of the local speed $\mathbf{u}(\mathbf{r} - \mathbf{e}_i\delta t, t)$; the local speed $\mathbf{u}(\mathbf{r} - \mathbf{e}_i\delta t, t)$ is necessary to calculate the equilibrium distribution function $f_i(\mathbf{r} - \mathbf{e}_i\delta t, t)$.

An equivalent procedure was already proposed in the LB litterature [5, 6, 7, 8]. This *Interpolation Supplemented Lattice Boltzmann* (ISLB) scheme (as named by their authors) has the same steps as described above, but their order is changed:

1. the collisions and the action of external forces are processed in the lattice nodes $\mathbf{r}$

2. the propagation step transports the result towards the points $\mathbf{r} + \mathbf{e}_i\delta t$

3. the interpolation step recovers the new distribution functions in the lattice node $\mathbf{r}$ at time $t + \delta t$

To avoid confusion with another scheme to be described below, we will denote this original ISLB scheme as ISLB-CP (ISLB Collision – Propagation) since the collision step is done before the propagation step. Figure 1.6 recalls the main characteristics of this ISLB-CP scheme. In this figure, lattice nodes are denoted by 0, 1, ... 8. Let $\mathbf{r}_0$, $\mathbf{r}_1$, $\ldots \mathbf{r}_8$ the position vectors of these lattice nodes. After processing the effect of collisions (as well as the effect

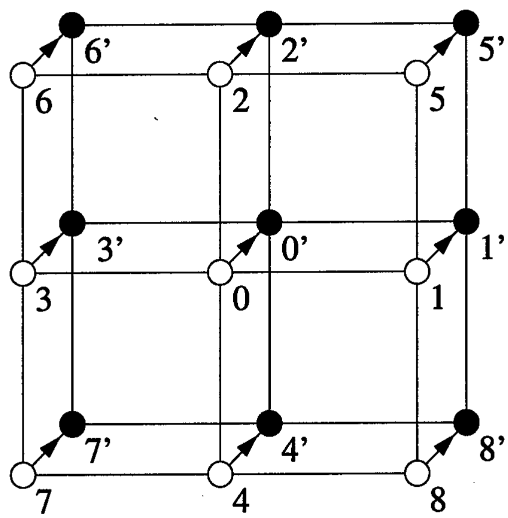


Figure 1.6: The ISLB-CP scheme.

of external forces) in the original lattice nodes, particle distribution functions are propagated along their characteristics lines during the time step δt and arrive in the displaced nodes $0'$, $1'$, $\ldots 8'$, whose position vectors are $\mathbf{r}'_0$, $\mathbf{r}'_1$, $\ldots \mathbf{r}'_8$ (figure 1.6 shows the case when the displacement is made along the $\mathbf{e}_5$ direction). Thus, for each vector $\mathbf{e}_i$, $i = 1, 2, \ldots 8$, the collision procedure, followed by the propagation procedure, transforms the initial set of distribution functions $\{f_i(\mathbf{r}_n, t)\}$, $n = 0, 1 \ldots 8$ into the set $\{f_i(\mathbf{r}'_n, t + \delta t)\}$, $n' = 0', 1', \ldots 8'$. A $2D$ interpolation procedure (to be described further) allows to recover the value of $f_i(\mathbf{r}_0, t + \delta t)$ [5, 6, 7] and thus, the automaton algorithm may be applied again for the next time step, and so on.

It is possible to imagine an ISLB scheme where propagation is done first, followed by collisions processing. This scheme is presented in figure 1.7. If we start from the Lattice Boltzmann Equations (1.11) we may write

$$\begin{aligned} \mathbf{x} &= \mathbf{r} - \mathbf{e}_i \delta t \\ f_i(\mathbf{x}, t) &= f_i(\mathbf{r}, t) - \delta t\, \mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r}, t) \end{aligned} \tag{1.29}$$

and thus

$$f_i(\mathbf{r}, t + \delta t) \simeq f_i(\mathbf{x}, t) - \frac{\delta t}{\tau_p} \left[f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t) \right] +$$

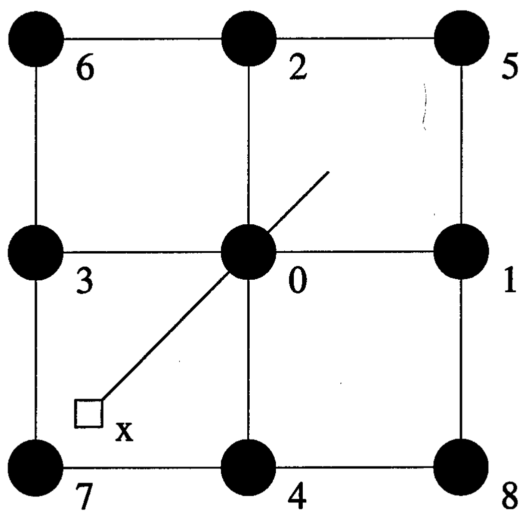


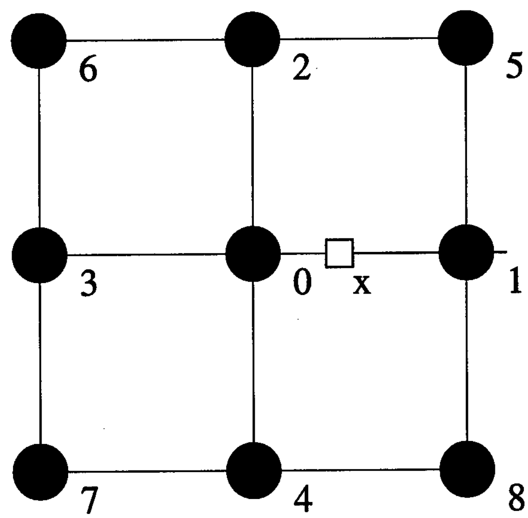
Figure 1.7: The ISLB-PC scheme.

$$\frac{\delta t}{k_B T}\,\mathbf{F}(\mathbf{x},t)\cdot[\,\mathbf{e}_i\,-\,\mathbf{u}(\mathbf{x},t)\,]\;f_i^{eq}(\mathbf{x},t)$$

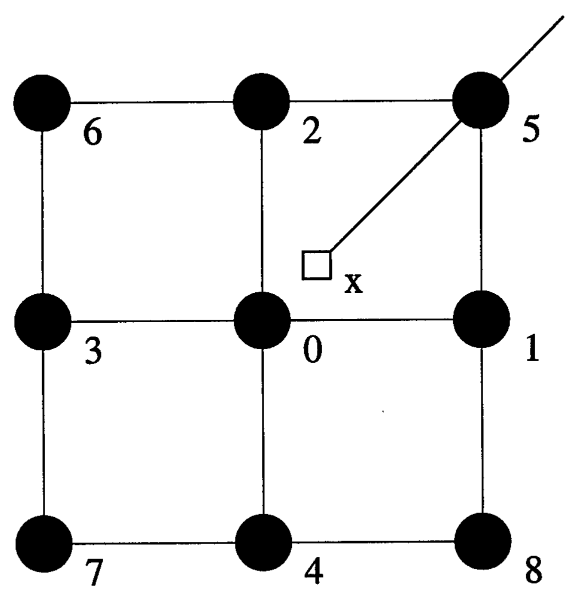
$$i\,=\,0,\,1,\,\ldots\,N \qquad (1.30)$$

In this scheme, which we will denote as ISLB-PC (since the Propagation step is done before the Collision step), an interpolation procedure in two dimensions is used to determine the value of $f_i(\mathbf{x},t)$ in Eq. (1.30). To find the value of $f_i(\mathbf{x},t)$ when one has the values $\{f_i(\mathbf{r}_n,t)\}$, $n\,=\,0,\,1\,\ldots\,8$, we may use the same interpolation procedure as for the ISLB-CP scheme, which gives $f_i(\mathbf{r}_0,t\,+\,\delta t)$ from the set $\{f_i(\mathbf{r}'_n,t\,+\,\delta t)\}$, $n'\,=\,0',\,1',\,\ldots\,8'$. The ISLB-PC approach may be viewed as being equivalent to the early Finite Difference schemes developed in [1], but replaces the calculation of local derivatives through an interpolation procedure, while maintaining the propagation of information along the characteristics line.

The ISLB-CP and ISLB-PC schemes use biquadratic interpolation to calculate the values of the distribution functions in the central node (denoted by 0 in Figures 1.6 and 1.7). This is not the unique possibility. In fact, the authors who introduced the original ISLB scheme [5, 6, 7, 8] reccommmended the use of an upwind biquadratic interpolation procedure. This procedure is explained briefly in Figure 1.8, where the position of the point $\mathbf{x}$ is shown



(a)



(b)

Figure 1.8: The UISLB-PC scheme.

in two cases, when the characteristics lines are directed along the velocity vectors $\mathbf{e}_5$ and $\mathbf{e}_1$, respectively (the other velocity directions may be considered in a similar way). In these Upwind ISLB-PC (UISLB-PC) cases, the distribution functions are interpolated in node x and thereafter propagated on the corresponding characteristics lines towards the lattice nodes 5 (Figure 1.8a) and 1 (Figure 1.8b). An Upwind ISLB-CP (UISLB-CP) scheme may be introduced in the same way as the UISLB-PC scheme described here.

The *Iterpolation Supplemented Lattice Boltzmann schemes* were originally developed to deal with non - uniform grids. To our best knowledge, in the LB litterature there is no attempt to use ISLB schemes to multiple component systems whose particles carry different masses. When developing the ISLB-PC scheme, their authors started from the early LGLB philosophy, which considers that distribution functions are redistributed at each lattice node because of collisions and are moved thereafter in the direction of their corresponding velocity vectors. When compared to this philosophy, the present discussion of Finite Difference Schemes for the Boltzmann Equation, which is based on the characteristics lines of the Boltzmann equation, may be another step towards a more rigorous approach to Lattice Boltzmann models. Because the Lattice Boltzmann equation is a hyperbolic equation, the characteristics curves (which are straight lines since the velocity vectors $\mathbf{e}_i$ are constant [4]) may provide the background for error analysis or stability investigations.

1.9.2 Interpolation procedures on the $2D$ lattice

Both ISLB-CP and ISLB-PC schemes need an interpolation procedure to evaluate the value of the distribution function $f_i(\mathbf{x}, t)\}$ at the point $\mathbf{x} = (x, y) = \mathbf{r}_0 - \mathbf{e}_i \delta t$ in figure 1.7, when the corresponding values $\{f_i(\mathbf{r}_n, t)\}$, $n = 0, 1, \ldots 8$ of the distribution function are known. As pointed in [8], second order interpolation is necessary in order to avoid spurious dependence of the fluid viscosity on the lattice size.

If we introduce a cartesian coordinate system centered in the node $\mathbf{r}_0 = (x_0, y_0)$ in figure 1.7, such that the X axis points along the vector $\mathbf{e}_1$ and we use the fact that we deal with a square lattice, we may define the non - dimensional (natural) coordinates [9]

$$\xi = \frac{x - x_0}{\delta x} \in [-1, +1]$$

$$\eta \;=\; \frac{y - y_0}{\delta x} \in [-1, +1] \tag{1.31}$$

To simplify the notation, in this subsection we will write $f(\mathbf{x})$ instead of $f_i(\mathbf{x}, t)$, and $f(\mathbf{r}_n)$, $n = 0, 1, \ldots 8$ instead of $f_i(\mathbf{r}_n, t)$. The interpolation procedure gives

$$f(\mathbf{x}) = \sum_{n=0}^{n=8} N_n \, f(\mathbf{r}_n) \tag{1.32}$$

where the weight coefficients N_n are given in table 1.1 as the product of two second order Lagrange interpolation coefficients (one for each axis), which are defined as follows (the argument θ stands for ξ or η)

$$\begin{aligned}
L_-^2(\theta) &= \frac{[\theta - 0][\theta - 1]}{[(-1) - 0][(-1) - 1]} \\
L_0^2(\theta) &= \frac{[\theta - (-1)][\theta - 1]}{[0 - (-1)][0 - 1]} \\
L_+^2(\theta) &= \frac{[\theta - (-1)][\theta - 1]}{[1 - (-1)][1 - 0]}
\end{aligned} \tag{1.33}$$

The biquadratic interpolation procedure described above may be applied to any lattice node which is not situated on the boundary (i.e., bulk nodes). Bulk nodes $\mathbf{r}_0$ have all the eight neighbors $\mathbf{r}_n$, $n = 0, 1, \ldots 8$. When the node $\mathbf{r}_0$ is located on the lattice boundary, some of these neighbors are missing, e.g., as shown in figure 1.9, which refers to four kinds of boundaries (walls): bottom (B), top (T), left (L) and right (R), Such situations are encountered when one deals with flow between horizontal or parallel plates (we will not discuss here the case when the fluid system is confined into a box – this case needs to consider also the presence of corner nodes).

For wall nodes, Eq. (1.32) changes to

$$f(\mathbf{x}) = \sum_{n \in \mathcal{N}_W} N_n \, f(\mathbf{x}_n) \tag{1.34}$$

where the sum is computed with respect to all indices in the set $\mathcal{N}_W$, $W \in \{B, T, L, R\}$. The elements of the sets $\mathcal{N}_B$ are (see figure 1.9):

$$\mathcal{N}_B \;\equiv\; \{0, 1, 2, 3, 5, 6\}$$

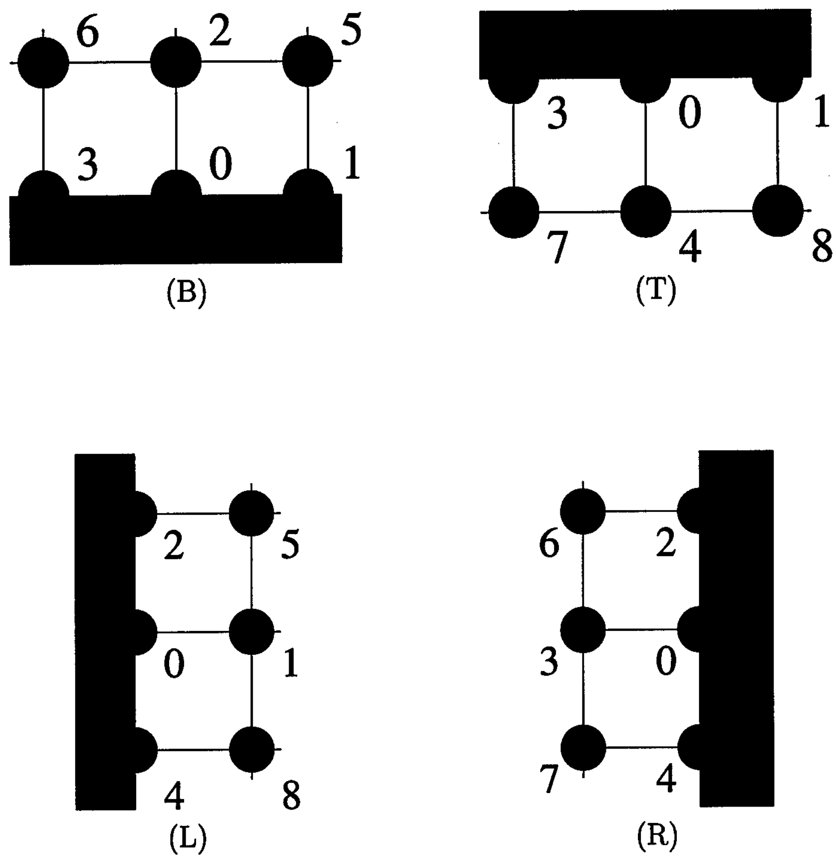


Figure 1.9: Lattice nodes on the walls: (B) – bottom wall; (T) – top wall; (L) – left wall; (R) – right wall.

Table 1.1: Interpolation weights for the bulk lattice nodes.

N_0	$L_0^2(\xi)\, L_0^2(\eta)$
N_1	$L_+^2(\xi)\, L_0^2(\eta)$
N_2	$L_0^2(\xi)\, L_+^2(\eta)$
N_3	$L_-^2(\xi)\, L_0^2(\eta)$
N_4	$L_0^2(\xi)\, L_-^2(\eta)$
N_5	$L_+^2(\xi)\, L_+^2(\eta)$
N_6	$L_-^2(\xi)\, L_+^2(\eta)$
N_7	$L_-^2(\xi)\, L_-^2(\eta)$
N_8	$L_+^2(\xi)\, L_-^2(\eta)$

$$\begin{aligned}
\mathcal{N}_T &\equiv \{0,1,3,4,7,8\} \\
\mathcal{N}_L &\equiv \{0,1,2,4,5,8\} \\
\mathcal{N}_R &\equiv \{0,2,3,4,6,7\}
\end{aligned}$$

We may use linear Lagrange interpolation for the Y directions when dealing with horizontal walls, or for the X direction when dealing with vertical walls. To this purpose, we define the corresponding coefficients

$$\begin{aligned}
L_0^{1-}(\theta) &= \frac{\theta - (-1)}{0 - (-1)} \\
L_-^{1-}(\theta) &= \frac{\theta - 0}{(-1) - 0} \\
L_0^{1+}(\theta) &= \frac{\theta - 1}{0 - 1} \\
L_+^{1-}(\theta) &= \frac{\theta - 0}{1 - 0} \qquad (1.35)
\end{aligned}$$

while the interpolation weights N_n are given in table 1.2, for each boundary.

Table 1.2: Interpolation weights for the boundary lattice nodes.

bottom	N_0	$L_0^2(\xi)\, L_0^{1+}(\eta)$
	N_1	$L_+^2(\xi)\, L_0^{1+}(\eta)$
	N_2	$L_0^2(\xi)\, L_+^{1+}(\eta)$
	N_3	$L_-^2(\xi)\, L_0^{1+}(\eta)$
	N_5	$L_+^2(\xi)\, L_+^{1+}(\eta)$
	N_6	$L_-^2(\xi)\, L_+^{1+}(\eta)$
top	N_0	$L_0^2(\xi)\, L_0^{1-}(\eta)$
	N_1	$L_+^2(\xi)\, L_0^{1-}(\eta)$
	N_3	$L_-^2(\xi)\, L_0^{1-}(\eta)$
	N_4	$L_+^2(\xi)\, L_-^{1-}(\eta)$
	N_7	$L_-^2(\xi)\, L_-^{1-}(\eta)$
	N_8	$L_0^2(\xi)\, L_-^{1-}(\eta)$
left	N_0	$L_0^0(\xi)\, L_0^2(\eta)$
	N_1	$L_+^{1+}(\xi)\, L_0^2(\eta)$
	N_2	$L_0^{1+}(\xi)\, L_+^2(\eta)$
	N_4	$L_0^{1+}(\xi)\, L_-^2(\eta)$
	N_5	$L_+^{1+}(\xi)\, L_+^2(\eta)$
	N_8	$L_+^{1+}(\xi)\, L_-^2(\eta)$
right	N_0	$L_0^{1-}(\xi)\, L_0^2(\eta)$
	N_2	$L_0^{1-}(\xi)\, L_+^2(\eta)$
	N_3	$L_-^{1-}(\xi)\, L_0^2(\eta)$
	N_4	$L_0^{1-}(\xi)\, L_-^2(\eta)$
	N_6	$L_-^{1-}(\xi)\, L_+^2(\eta)$
	N_7	$L_-^{1-}(\xi)\, L_-^2(\eta)$

We should underline the fact that the ISLB schemes (ISLB-CP, ISLB-PC, UISLB-CP and UISLB-PC), which use second order (biquadratic) $2D$ interpolation procedures, are different from the previously introduced Upwind, Lax - Friedrichs, Space Centered and Lax - Wendroff schemes, which use $1D$ interpolation. We want to point that other $2D$ interpolation schemes may be also considered, besides the ISLB schemes (i.e., biquadratic interpolation procedure adopted here for bulk nodes, associated with a linear - quadratic procedure for boundary nodes). For example, the bilinear Lagrange interpolation, as well as the linear or the quadratic elements in the "serendipity family" [9] may provide some alternatives.

1.10 Numerical simulations

1.10.1 Computer code

In the remaining section of this chapter, we are reporting a few significant simulation results which were obtained using the explicit Finite Difference Lattice Boltzmann (FDLB) schemes introduced above. These results refer to $2D$ Poisseuille flow and were mainly done to see the behavior and to estimate the errors and performances of each scheme we investigated.

To reduce the programming effort, we restricted ourserves to the nine bit model and used a square lattice. The computer code is given in Appendix A. All the necessary simulation parameters (number of lattice nodes in the X and Y direction, number of cycles to be performed, numerical scheme to be used, boundaries, initial configuration, particle masses, local number densities, wall velocities and so on) are introduced in the input data file `wet9.input`. The input parameter `key_scheme` controls the numerical scheme to be used during the simulation, as shown in Table 1.3. For further details concerning the meaning of input parameters in the file `wet9.input` please refer to the comment lines in the source code.

The other chapters of our present report will be dedicated to the physics of diffusion, surface tension and wetting phenomena, as it can be recovered using the computer codes based on the explicit numerical schemes we developed.

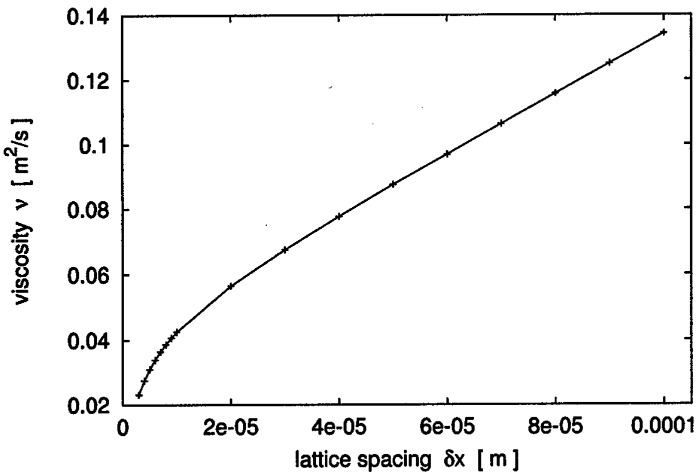
1.10.2 Single Component Fluid

We made different computer runs to study the influence of the parameters δx, δt and τ_p on the kinematic viscosity ν. Initially, the flow of a single component fluid within a $2D$ channel (Poisseuile flow) was simulated using the Upwind, Lax - Friedrichs and Lax - Wendroff schemes. A 5×25 lattice was used, with the same forcing term. The value of the viscosity was determined after fitting the parabolic velocity profile across the channel, as done in our intermediate report [1].

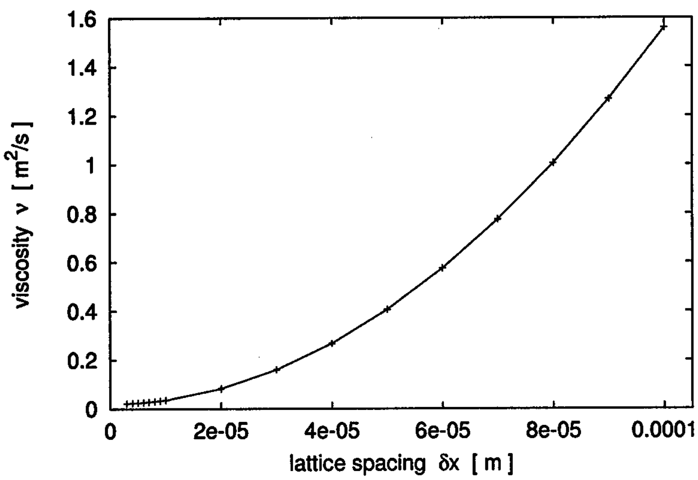
Figure 1.10 shows the influence of the lattice spacing δx on the kinematic viscosity ν, when the other parameters are constant ($\tau_p = 10^{-8}$ s, $\delta t = 10^{-9}$ s, particle mass $m = 1$ amu, temperature $T = 300$ K, force $F = 10^{-22}$ N). When using the first order schemes (Upwind, Lax - Friedrichs or FU), the viscosity is always found to increase when the lattice spacing δx increases. In particular, when using the Upwind or the FU scheme, a linear dependence of

Table 1.3: Possible values of the input parameter `key_scheme`, which controls the numerical scheme to be used during computer simulations.

`key_scheme`	Numerical Scheme	Section
0	Upwind (U)	1.5
1	Lax – Friedrichs (LF)	1.6
2	Space Centered (SC)	1.7
3	Lax – Wendroff (LW)	1.8
7	ISLB – CP	1.9
8	UISLB – CP	1.9
9	ISLB – PC	1.9
10	UISLB – PC	1.9
11	Linear Serendip Elements	1.9
12	Bilinear Interpolation	1.9
13	First Order Upwind (FU)	1.2
14	Centered Space (CS)	1.2
15	Second Order Upwind (SU)	1.2

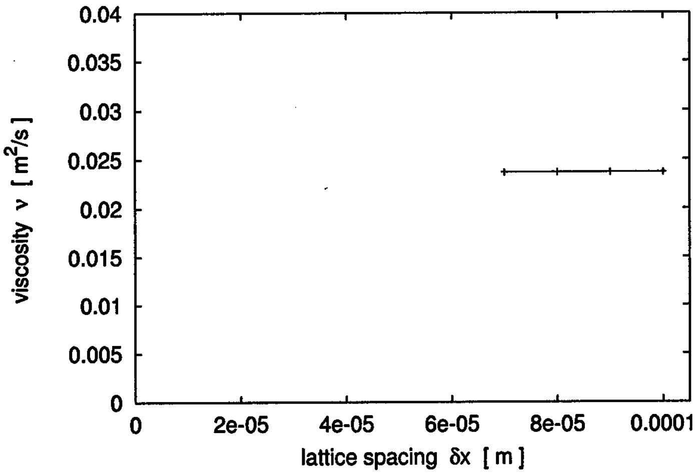


(a)

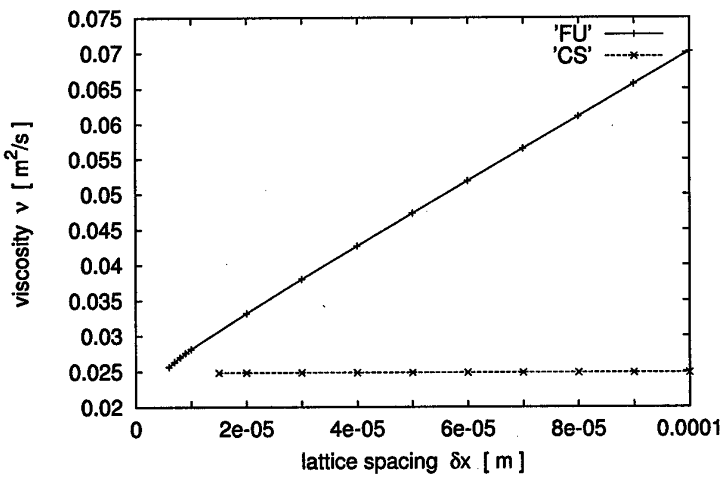


(b)

Figure 1.10: Influence of the lattice spacing δx on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.



(c)



(d)

Figure 1.10: *(continued)* Influence of the lattice spacing δx on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

the viscosity on the lattice spacing δx is found when this parameter becomes large enough. The kinematic viscosity ν is found to be independent on the lattice spacing only when using second order schemes (Lax - Wendroff or CS). Similar results were reported when using the ISLB scheme [5, 6]. A rigorous analysis of an $1D$ ISLB model [8], revealed that one should use a second order interpolation formula in order to avoid a spurious (lattice spacing dependent) viscosity term in the Navier Stokes equation.

Despite the recovery of the correct (lattice spacing independent) value of the viscosity, the Lax Wendroff scheme in our computer code is found to become unstable when the Courant - Friedrichs - Lewy number exceeds a certain value ($CFL \simeq 0.04$). This is why the values $\nu = \nu(\delta x)$ in figure 1.10c are available only for $\delta x \geq 7 \times 10^{-5}$ m. This behavior may be associated to the necessity to keep a small value of the Courant - Friedrichs - Lewy number CFL, in order to preserve the validity of the first order series expansion (1.29), which is essential to all interpolation based FDLB scheme. A similar problem was observed when using the CS scheme (Figure 1.10d) but, in this case, the lattice spacing δx should be larger than 1.5×10^{-5} m (which means $CFL \simeq 0.2$) in order to avoid the negative values of the equilibrium distribution functions.

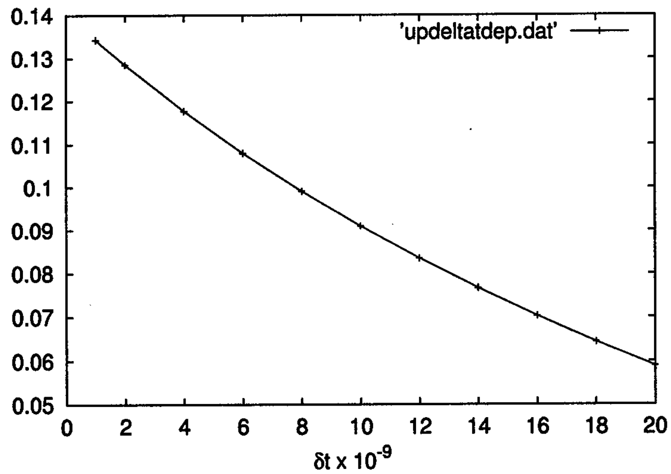
Figure 1.11 shows the influence of the time step δt on the kinematic viscosity, when $\delta x = 10^{-4}$ m and the other parameters were unchanged. When using the Upwind, Lax - Friedrichs or Lax - Wendroff schemes, the viscosity ν is always found to decrease when increasing the time step. Only in the case of the Lax - Wendroff scheme, the dependence $\nu = \nu(\delta t)$ is a linear one, in accordance to the general theory of the LGLB model [1, 10, 11]

$$\nu = \frac{2\tau_p - \delta t}{2} \chi c^2 = \frac{2\tau_p - \delta t}{2} \frac{k_B T}{m} \tag{1.36}$$

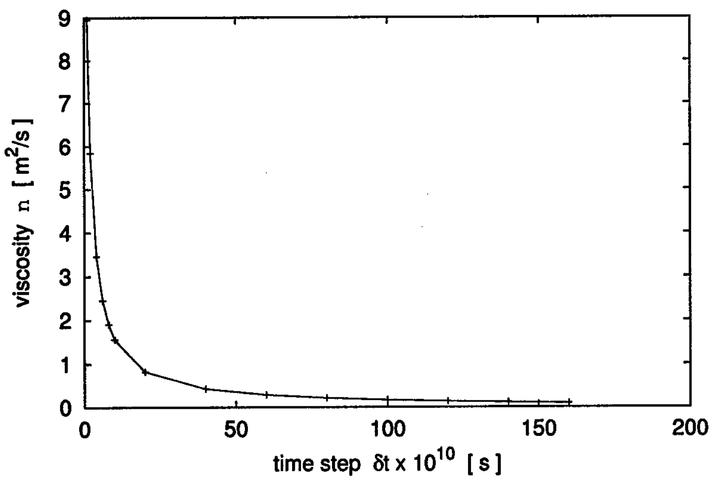
The viscosity is found to be independent on the time step when using the Finite Difference Schemes (FU and CS). This is in accordance to the viscosity formula derived previously in our intermediate report [1]

$$\nu = \tau_p \chi c^2 = \frac{\tau_p}{m} k_B T \tag{1.37}$$

Figure 1.12 shows the influence of the relaxation time τ_p, when $\delta x = 10^{-4}$ m and $\delta t = 10^{-9}$ s. Eq. (1.36), which predicts a linear dependence of the viscosity on the relaxation time τ_p, is found again to be valid for the Lax - Wendroff scheme, while Eq. (1.37) is well verified in the case of the Centered

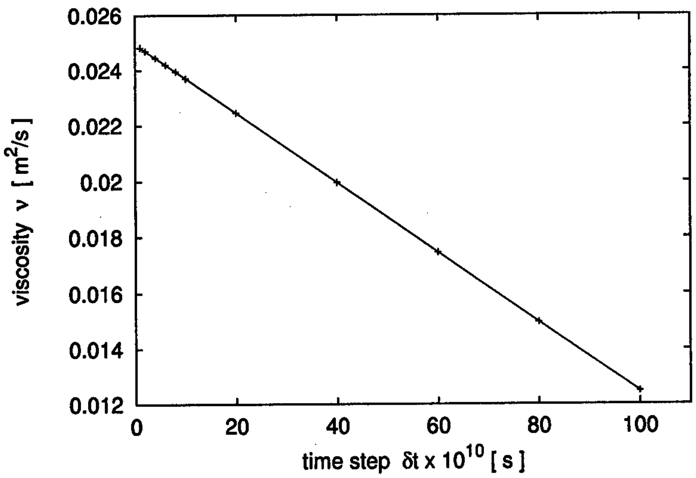


(a)

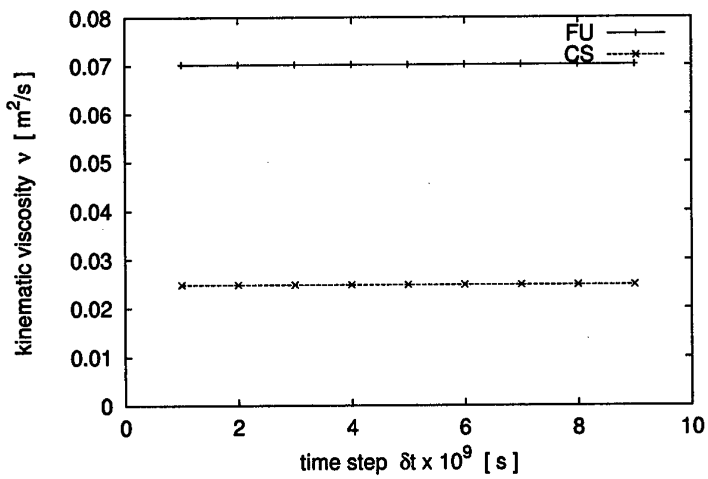


(b)

Figure 1.11: Influence of the time step δt on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

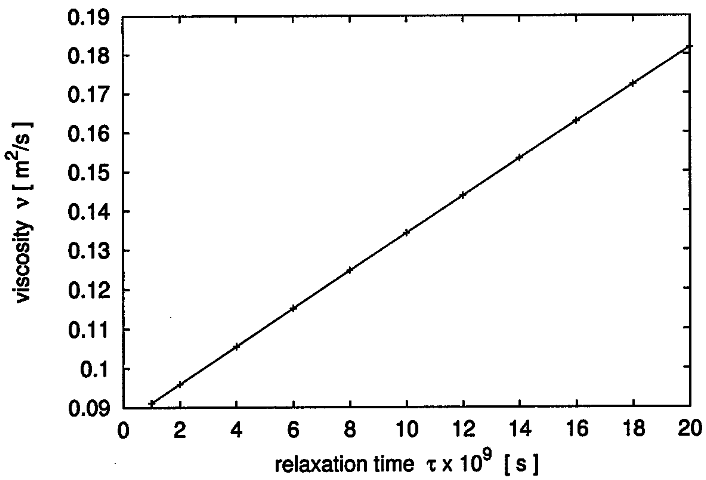


(c)

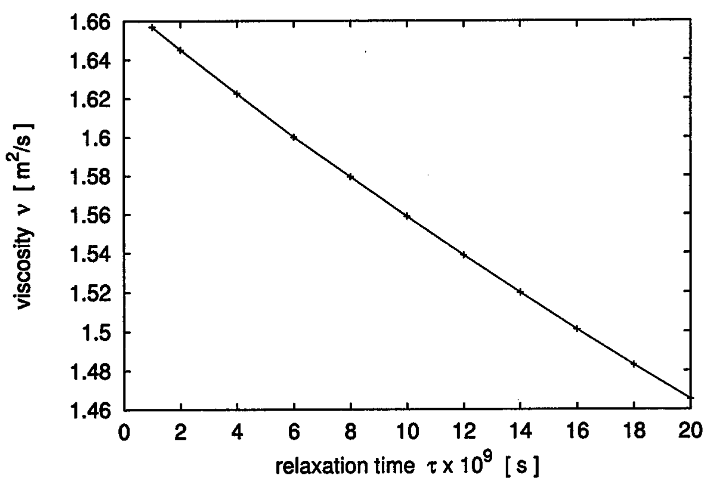


(d)

Figure 1.11: *(continued)* Influence of the time step δt on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

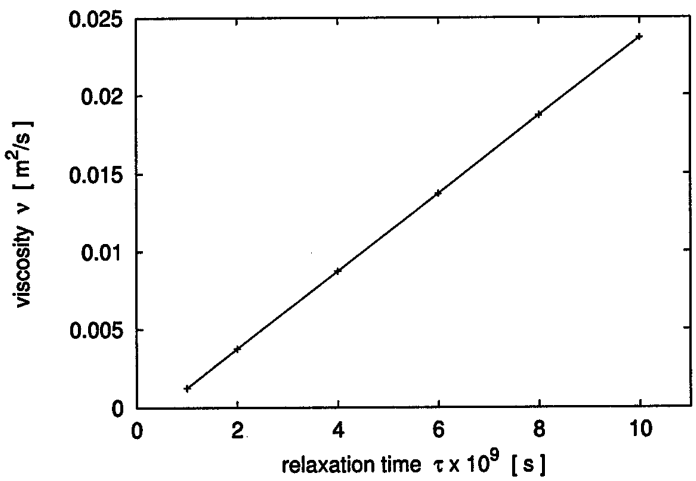


(a)

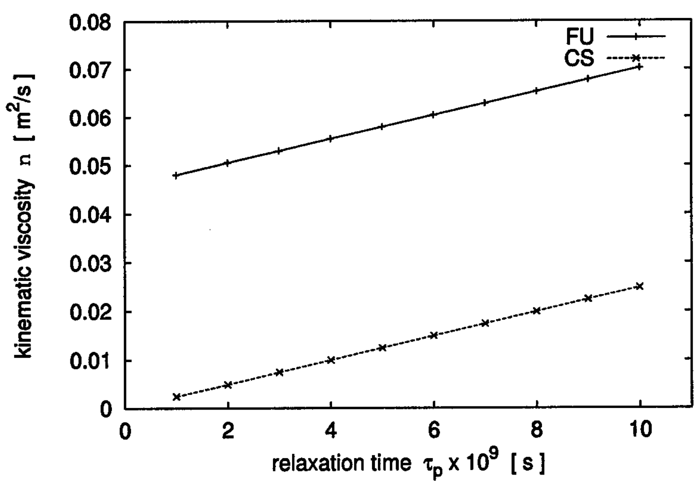


(b)

Figure 1.12: Influence of the relaxation time τ_p on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme.



(c)



(d)

Figure 1.12: *(continued)* Influence of the relaxation time τ_p on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

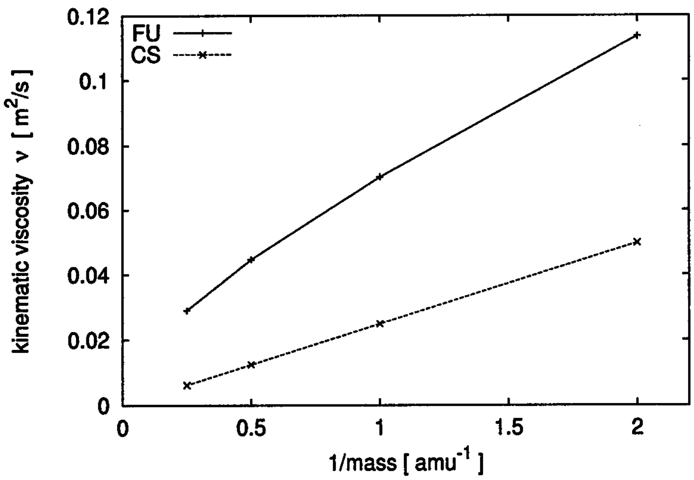


Figure 1.13: Dependence of the kinematic viscosity ν on the inverse of the mass (m^{-1}) of the particles in the fluid system, as determined by fitting the parabolic velocity profile in a $2D$ channel (FU and CS schemes).

Space (CS) scheme. Even if the Upwind and the First Order Upwind (FU) schemes give also a linear dependence $\nu = \nu(\tau_p)$, the values of the viscosity are not within the range predicted by Eqs. (1.36) and (1.37), respectively.

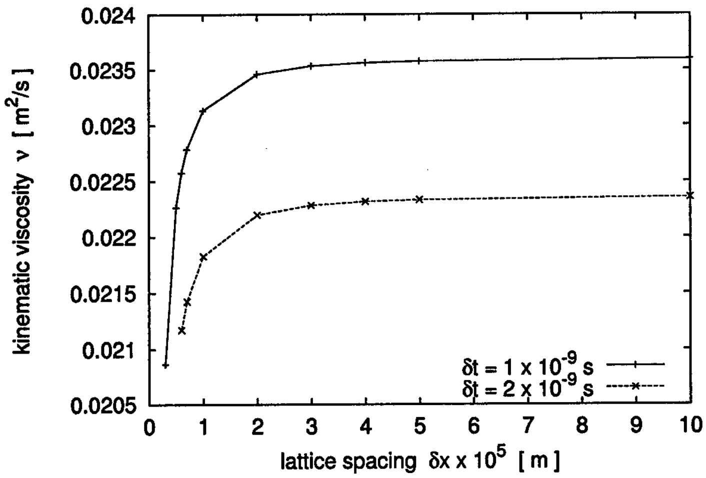
Although the Lax - Wendroff scheme gives the correct dependence of the viscosity on the simulation parameters δx, δt and τ_p, we found it to be unstable when dealing with a homogeneous two component fluid system. It is this reason why we turned later to other schemes, which proved good stability for such systems.

To check the FU and CS schemes further, we performed several runs with $\delta x = 0.0001$ m, $\delta t = 10^{-9}$ s, $\tau_p = 10^{-8}$ s and different values of the mass m of the particles. The results are shown in Figure 1.13. A linear dependence of the kinematic viscosity on the inverse of the mass (m^{-1}), which is predicted by Eq. (1.37) is found only in the case of the CS scheme.

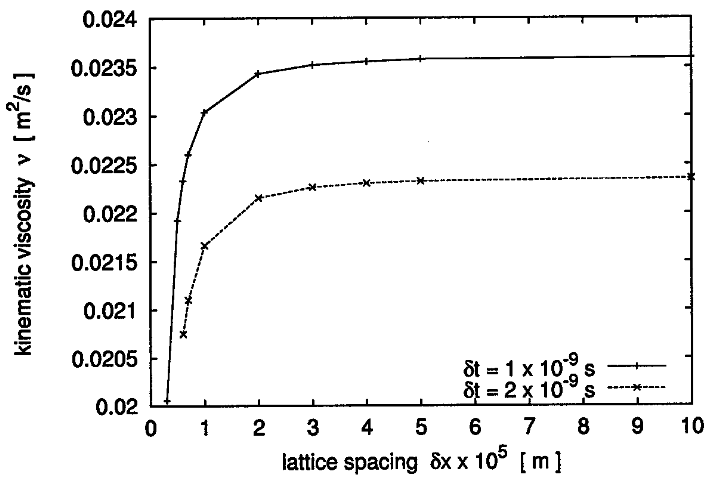
Table 1.4 and Figure (1.14) show the dependence of the kinematic viscosity ν on the lattice spacing δx when centered ISLB schemes are used. Two values of the time step δt were adopted, while the relaxation time was maintained constant ($\tau_p = 1 \times 10^{-8}$ s). One may see that table 1.4 has a few missing entries in the column $\delta t = 2 \times 10^{-9}$ s. These missing en-

Table 1.4: Influence of the lattice spacing δx on the viscosity (ISLB schemes).

	δx [s]	ν [m^2/s]	ν [m^2/s]
		$\delta t = 1 \times 10^{-9}$ s	$\delta t = 2 \times 10^{-9}$ s
ISLB-CP	0.3×10^{-5}	0.020865	
	0.5×10^{-5}	0.022264	
	0.6×10^{-5}	0.022575	0.021173
	0.7×10^{-5}	0.022785	0.021425
	1.0×10^{-5}	0.023132	0.021829
	2.0×10^{-5}	0.023457	0.022198
	3.0×10^{-5}	0.023532	0.022284
	4.0×10^{-5}	0.023560	0.022317
	5.0×10^{-5}	0.023573	0.022332
	10.0×10^{-5}	0.023592	0.022354
ISLB-PC	0.3×10^{-5}	0.020059	
	0.5×10^{-5}	0.021926	
	0.6×10^{-5}	0.022332	0.020750
	0.7×10^{-5}	0.022603	0.021105
	1.0×10^{-5}	0.023093	0.021665
	2.0×10^{-5}	0.023433	0.022155
	3.0×10^{-5}	0.023521	0.022265
	4.0×10^{-5}	0.023554	0.022306
	5.0×10^{-5}	0.023577	0.022326
	10.0×10^{-5}	0.023592	0.022352



(a)



(b)

Figure 1.14: Influence of the lattice spacing δx on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel, for two values of the time step δt (1×10^{-9} s and 2×10^{-9}: (a) - ISLB-CP scheme; (b) - ISLB-PC scheme.

tries correspond to CFL > 1, when the numerical scheme is always unstable and distribution functions become negative. For both $\delta t = 1 \times 10^{-9}$ s and $\delta t = 2 \times 10^{-9}$ s, the kinematic viscosity increases when increasing the lattice spacing δx (i.e., when decreasing the Courant - Friedrichs - Lewy number CFL below unity). When the lattice spacing is large enough to achieve CFL < 0.1 (approximatively), the kinematic viscosity becomes very close to the theoretical value calculated in accordance to Eq. (1.36). This feature is seen clearly in Figure 1.14, where the curves $\nu = \nu(\delta x)$ exhibit a saturation behavior. Thus, the error in the determination of the viscosity becomes negligible (less that one percent in our case) when the first order series expansion (1.29) becomes accurate enough.

In order to see the influence of the time step δt on the viscosity, we performed several computer runs with the ISLB schemes, using a fixed value of the lattice spacing δx. This value was choosen to be $\delta x = 3.0 \times 10^{-5}$ m, which lies within the interval where the approximation (1.29) is acceptable. The results, which are reproduced in Table 1.5, as well as in Figure (1.15, show a linear dependence $\nu = \nu(\delta t)$, as expected in accordance to Eq. (evisco). This equation predicts also a linear dependence of the kinematic viscosity ν with respect to the relaxation time τ_p, which is seen in Figure 1.16, for $\delta x = 3.0 \times 10^{-5}$ m and $\delta_t = 1.0 \times 10^{-9}$ s

The numerical experiments described in this subsection revealed that only the CS scheme, as well as the ISLB-CP and ISLB-PC schemes give the correct dependence of the fluid viscosity on the simulation parameters, for a single component fluid.

Table 1.5: Influence of the time step δt on the kinematic viscosity ν (ISLB schemes).

	δt [s]	ν [m^2/s]
ISLB-CP	0.5×10^{-9}	0.024165
	1.0×10^{-9}	0.235323
	1.5×10^{-9}	0.022906
	2.0×10^{-9}	0.022284
	2.5×10^{-9}	0.021665
	3.0×10^{-9}	0.021047
ISLB-PC	0.5×10^{-9}	0.024159
	1.0×10^{-9}	0.023521
	1.5×10^{-9}	0.022891
	2.0×10^{-9}	0.022265
	2.5×10^{-9}	0.021642
	3.0×10^{-9}	0.021022

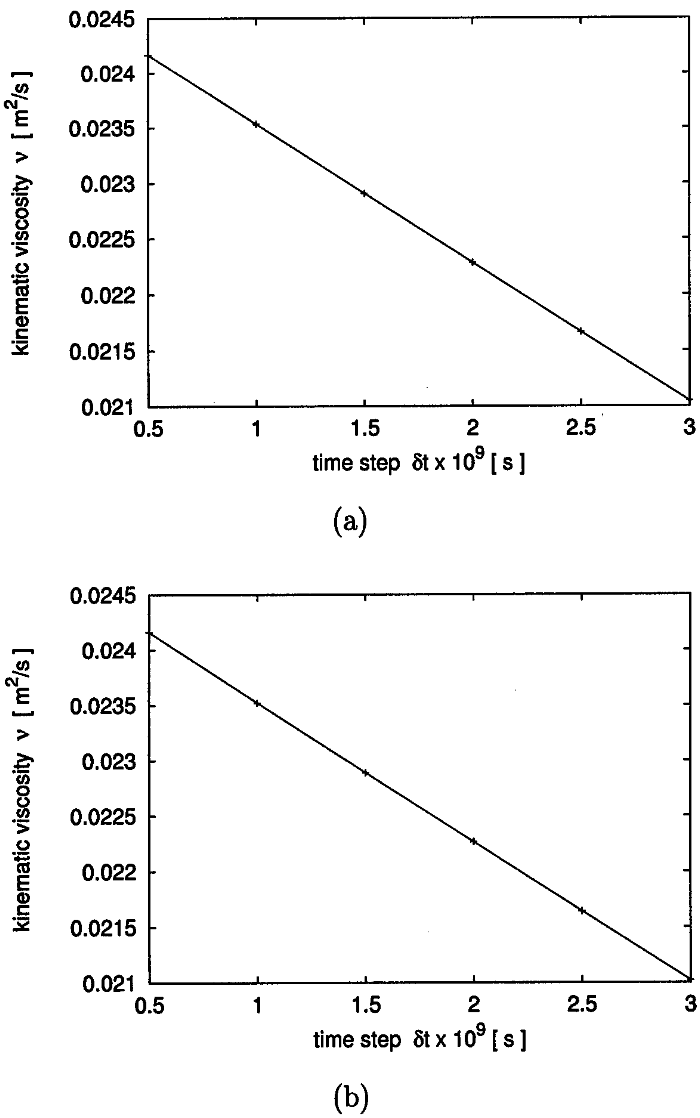
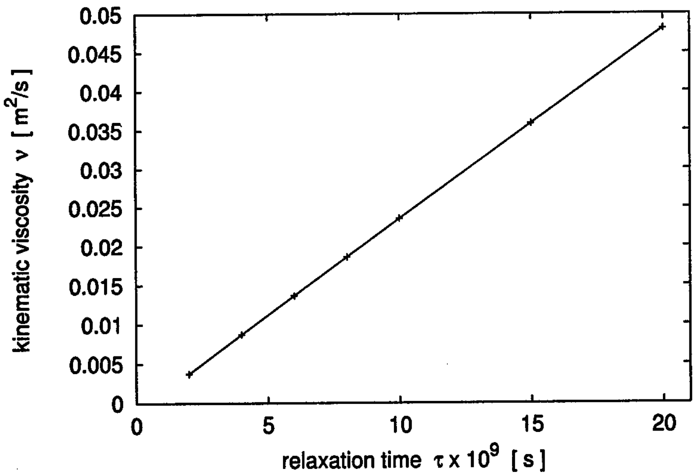
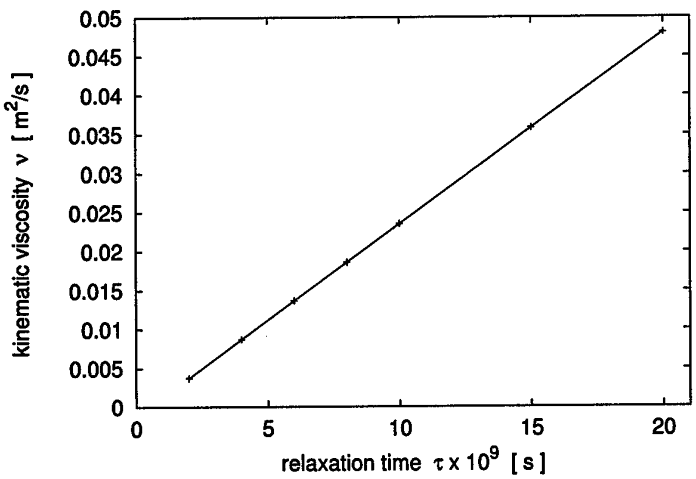


Figure 1.15: Influence of the time step δt on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel: (a) - ISLB-CP scheme; (b) - ISLB-PC scheme.



(a)



(b)

Figure 1.16: Influence of the relaxation time τ_p on the kinematic viscosity ν, as determined by fitting the parabolic velocity profile in a $2D$ channel, for a fixed of the time step δt (1×10^{-9} s): (a) - ISLB-CP scheme; (b) - ISLB-PC scheme.

1.10.3 Homogeneous Two Component Fluid

We consider a homogeneous two component fluid system subjected to Poiseuille flow, whose composition may be varied. The system's composition may be described using the mass composition

$$\omega = \frac{\rho_1}{\rho_1 + \rho_2} = \frac{\rho_1}{\rho} \tag{1.38}$$

where ρ_1 and ρ_2 are the mass densities of the two components and $\rho = \rho_1 + \rho_2$ is the mass density of the whole system. Another possibility to account for the composition of the homogeneous system is to use the mole fraction

$$x = \frac{n_1}{n_1 + n_2} = \frac{n_1}{n} \tag{1.39}$$

where n_1 and n_2 are the number of moles belonging to each component and $n = n_1 + n_2$ is the total number of moles in the system.

In order to study the composition dependence of the system's viscosity, we kept a constant number of moles $n = 1$ in the system during our simulations, while the number of moles belonging to the first component was varied from 0 to 1, with the increment 0.1. The system's viscosity was determined after fitting the parabolic velocity profile established across the channel, as done for the single component fluid.

The dynamic viscosity η of the system should equal the sum of the dynamic viscosities η_1 and η_2 of the two components

$$\eta = \eta_1 + \eta_2 \tag{1.40}$$

Since the dynamic viscosity η of a fluid is expressed as the product of its mass density ρ and kinematic viscosity ν

$$\eta = \rho\nu \tag{1.41}$$

we have, for the two component fluid system

$$\rho\nu = (\rho_1 + \rho_2)\nu = \rho_1\nu_1 + \rho_2\nu_2 \tag{1.42}$$

This gives a linear dependence of the kinematic viscosity ν of the whole system on the mass composition ω

$$\nu = \frac{\rho_1}{\rho}\nu_1 + \frac{\rho_2}{\rho}\nu_2 = \omega\,(\nu_1 - \nu_2) + \nu_2 \tag{1.43}$$

Thus, the slope of the straight line $\nu = \nu(\omega)$ is determined solely by the difference between the kinematic viscosities of each pure component. This slope is positive when $\nu_1 > \nu_2$ and negative otherwise.

Because the kinematic viscosity ν has a qualitatively different dependence on the relaxation time τ_p when using the Upwind or the Lax - Friedrichs schemes (see Figure 1.12), also the composition dependence of the viscosity of a binary system is found to have a different behavior when using these schemes. This is seen in Figure 1.17, where we show the results obtained for a binary system whose particles have the same mass, but the relaxation times τ_p are different for each component. Although the linear interpolation schemes (Upwind and Lax - Friedrichs schemes) do not give the correct dependence of the kinematic viscosity (1.36), the mass concentration dependence of the system viscosity is always linear, as predicted by Eq. (1.43) and shown in Figure 1.18.

Figures 1.19 and 1.20 show some results obtained with the ISLB-CP scheme. Similar graphs were obtained using the ISLB-PC scheme. The correct dependence of the viscosity of the whole system on the mass composition, as well as the other simulation parameters (masses m_1 and m_2 of the particles belonfing to each component, time step δt as well as relaxation times τ_1 and τ_2 of each component), is always recovered.

We should point here that the former LGLB model cannot account for results similar to those shown in figure 1.19 for a homogeneous two component system. In the LGLB model, all particles share the same thermal velocity, even if their masses are different. Since the viscosity in the LGLB model is mass independent, the use of this model to simulate a two component fluid system will always give a composition independent viscosity. This erroneous behavior of the LGLB model is limiting seriously its application to multicomponent fluid systems.

To compare the CS and the ISLB-CP schemes, we used the lattice spacing $\delta x = 0.0001$ m and the time step $\delta t = 10^{-9}$ s to study the composition dependence of the viscosity of a very particular two component system. The two components of this system had the masses $m_1 = 1$ amu, $m_2 = 2$ amu and the relaxation times $\tau_1 = 1 \times 10^{-8}$ s, $\tau_2 = 2 \times 10^{-8}$ s, respectively. In accordance to the viscosity formula (1.37), the viscosities of the two component fluids are identical when we use the CS scheme, since, for this particular system, we have

$$\frac{\tau_1}{m_1} = \frac{\tau_2}{m_2} \tag{1.44}$$

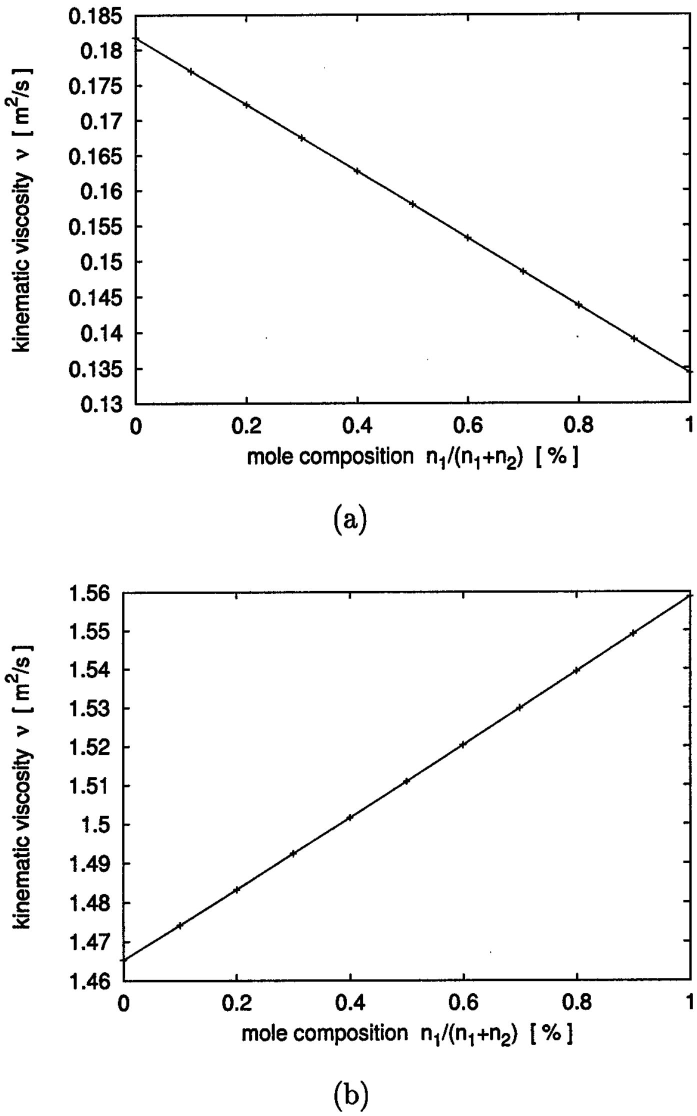
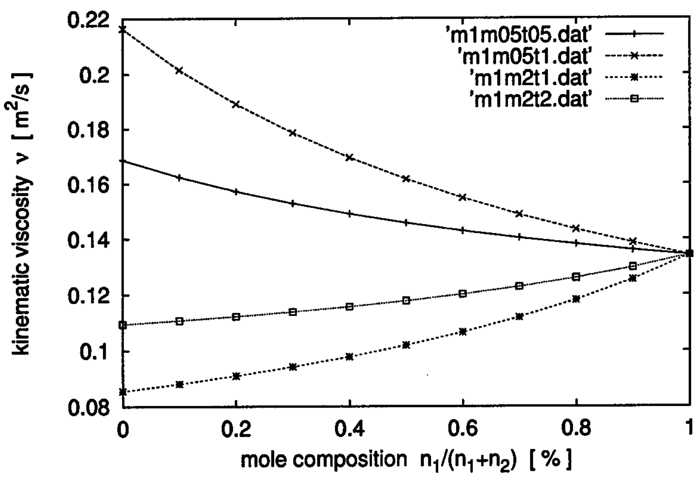
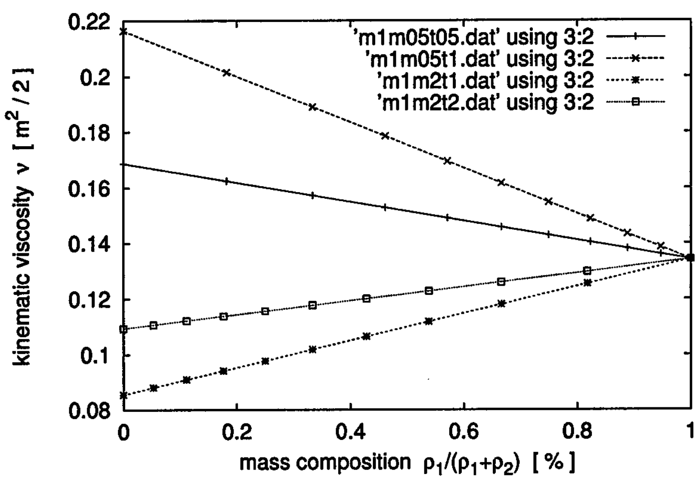


Figure 1.17: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme. The two components have the same mass (1 amu) but the relaxation times are different ($\tau_1 = 10^{-8}$ s and $\tau_2 = 2 \times 10^{-8}$ s).



(a)



(b)

Figure 1.18: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (Upwind scheme). The two components have different masses (expressed in amu) and/or relaxation times (expressed in s). Curves were been obtained for $m_1 = 1$, $m_2 = 0.5$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 0.5 \times 10^{-8}$, (upper curve), $m_1 = 1$, $m_2 = 0.5$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 1 \times 10^{-8}$, (second curve), $m_1 = 1$, $m_2 = 2$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 1 \times 10^{-8}$, (third curve), $m_1 = 1$, $m_2 = 2$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 2 \times 10^{-8}$, (bottom curve).

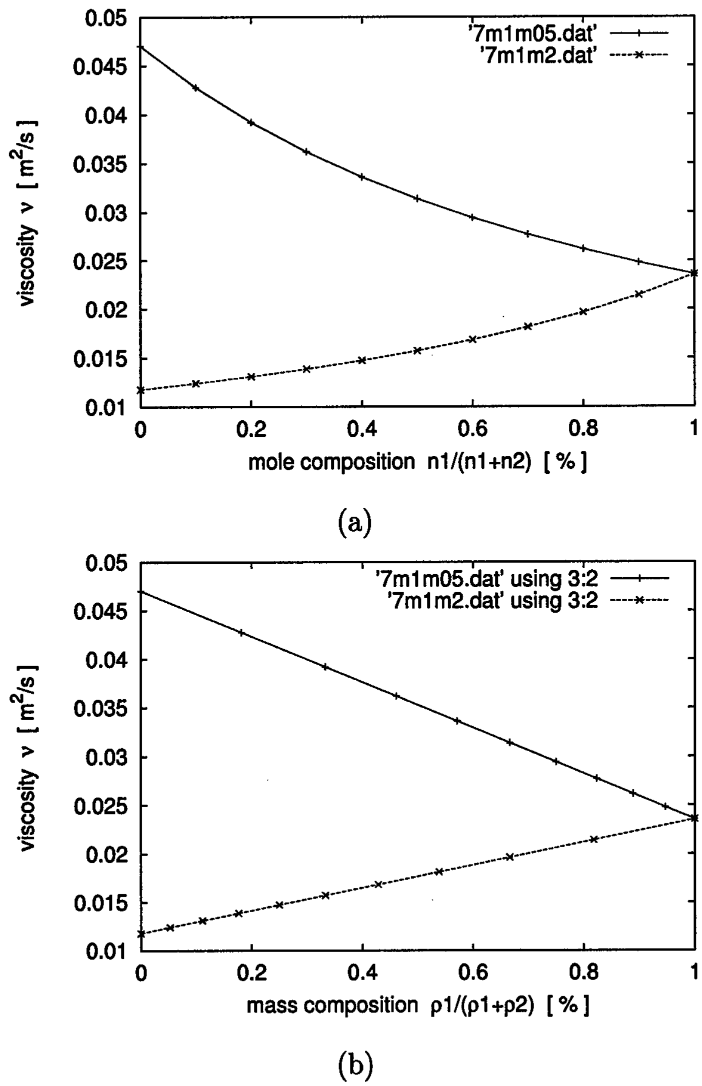


Figure 1.19: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (ISLB-CP scheme). The two components have different masses (expressed in amu) and the same relaxation times (expressed in s). Curves were been obtained for $m_1 = 1$, $m_2 = 0.5$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 1 \times 10^{-8}$, (upper curve) and $m_1 = 1$, $m_2 = 2$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 1 \times 10^{-8}$, (lower curve), respectively.

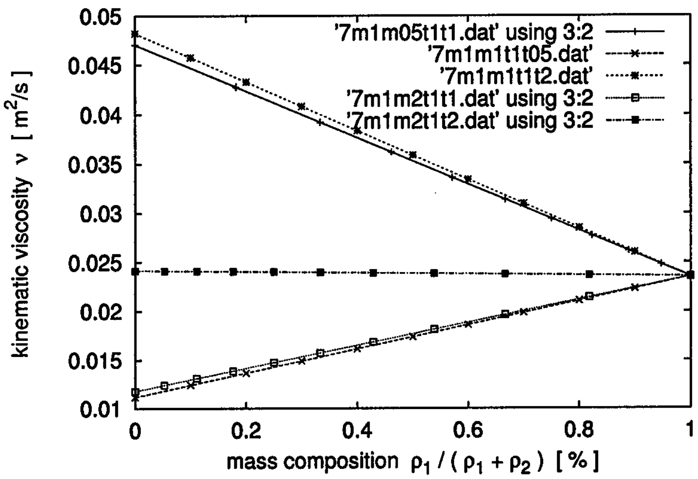


Figure 1.20: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (ISLB-CP scheme). Curves (from top to bottom) were been obtained for (masses are expressed in amu, while relaxation times are expressed in seconds): $m_1 = 1$, $m_2 = 1$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 2 \times 10^{-8}$, $m_1 = 1$, $m_2 = 0.5$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 1 \times 10^{-8}$, $m_1 = 1$, $m_2 = 2$, $\tau 1 = 1 \times 10^{-8}$, $\tau 2 = 2 \times 10^{-8}$, $m_1 = 1$, $m_2 = 1$, $\tau 1 = 1 \times 10^{-8}$, $\tau 2 = 0.5 \times 10^{-8}$, $m_1 = 1$, $m_2 = 2$, $\tau_1 = 1 \times 10^{-8}$, $\tau_2 = 2 \times 10^{-8}$.

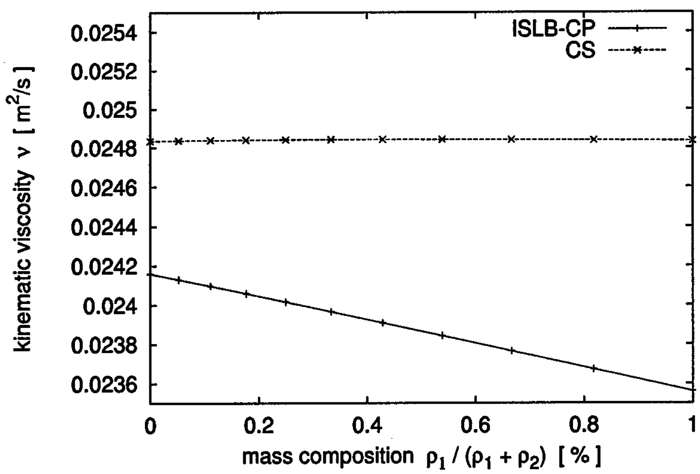


Figure 1.21: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (ISLB-CP and CS schemes). Curves were been obtained for $m_1 = 1$ amu, $m_2 = 2$ amu, $\tau_1 = 1 \times 10^{-8}$ s, $\tau_2 = 2 \times 10^{-8}$ s.

When we use the ISLB-CP scheme (as well as the ISLB-PC scheme), the viscosities of the two fluid components are no more equal since

$$\frac{2\tau_1 - \delta t}{m_1} \neq \frac{2\tau_2 - \delta t}{m_2} \tag{1.45}$$

Thus, the viscosity of the particular homogeneous two component system mentioned above is composition independent when the CS scheme is used, while it does not have this property when the ISLB schemes (ISPB-CP or ISLB-PC) are used instead. Figure 1.21 shows the computer results.

Chapter 2

Diffusion Couple

2.1 Description of the model

We consider a two component system whose particles carry different masses denoted m_σ, $\sigma = 1, 2$. The system is described by two distribution functions sets $\{ f_i^{\sigma,N}(\mathbf{r},t) \}$, $\sigma = 1, 2$; $i = 0, 1 \ldots N$, defined on a $2D$ regular lattice. The distribution function $f_i^{\sigma,N}(\mathbf{r},t)$ gives the probability to find in node $\mathbf{r}$ of the lattice, at time t, a particle of mass m_σ having the velocity $\mathbf{e}_i^{\sigma,N}$, $i = 0, 1 \ldots N$. As discussed previously [1], the underlying lattice may be a square one, while the velocity space may be restricted to one of the two $2D$ discrete velocity sets $\{ \mathbf{e}_i^{\sigma,N} \}$ widely used in LB simulations, which correspond to $N = 6$ or $N = 8$. These velocity sets define the so-called seven bit LB model ($N = 6$)

$$\begin{aligned} \mathbf{e}_0^{\sigma,6} &= 0 \\ \mathbf{e}_i^{\sigma,6} &\equiv \left(e_{i1}^{\sigma,6} \, , e_{i2}^{\sigma,6} \right) = \left[\cos \frac{2\pi(i-1)}{6} \, , \sin \frac{2\pi(i-1)}{6} \right] c_{\sigma,6} \\ & \quad i = 1, \ldots 6 \end{aligned} \tag{2.1}$$

or the nine bit model ($N = 8$)

$$\begin{aligned} \mathbf{e}_0^{\sigma,8} &= 0 \\ \mathbf{e}_i^{\sigma,8} &= \left[\cos \frac{\pi(i-1)}{2} \, , \sin \frac{\pi(i-1)}{2} \right] c_{\sigma,8} \quad , \quad i = 1, \ldots 4 \end{aligned} \tag{2.2}$$

$$\mathbf{e}_i^{\sigma,8} = \sqrt{2}\left[\cos\left(\frac{\pi}{4} + \frac{\pi(i-5)}{2}\right), \sin\left(\frac{\pi}{4} + \frac{\pi(i-5)}{2}\right)\right] c_{\sigma,8}$$

$$i = 5, \dots 8$$

The "thermal" velocities $c_{\sigma,N}$, $N \in \{6, 8\}$ are defined by

$$c_{\sigma,N} = \sqrt{\frac{k_B T}{m_\sigma \chi_N}} \tag{2.3}$$

where

$$\chi_N = \begin{cases} \frac{1}{4} & , \quad N = 6 \\ \frac{1}{3} & , \quad N = 8 \end{cases} \tag{2.4}$$

In the absence of external forces, the distribution functions are supposed to satisfy the Boltzmann equation with the Bhatnagar - Gross - Krook collision term ($\partial_t = \partial/\partial t$, $\partial_\alpha = \partial/\partial x_\alpha$, $\alpha = 1, 2$)

$$\partial_t f_i^{\sigma,N}(\mathbf{r},t) + e_{i\alpha}^{\sigma,N} \partial_\alpha f_i^{\sigma,N}(\mathbf{r},t) = -\frac{1}{\tau_\sigma}\left[f_i^{\sigma,N}(\mathbf{r},t) - f_i^{\sigma,N,eq}(\mathbf{r},t)\right]$$

$$\sigma = 1, 2 \qquad \text{and} \qquad i = 0, 1, \dots N \tag{2.5}$$

and therefore evolve in accordance to the Finite Difference Lattice Boltzmann (FDLB) equations

$$\begin{aligned} f_i^{\sigma,N}(\mathbf{r}, t+\delta t) &\simeq f_i^{\sigma,N}(\mathbf{r},t) - \delta t\, \mathbf{e}_i^{\sigma,N} \cdot \nabla_{\mathbf{r}} f_i^{\sigma,N}(\mathbf{r},t) \\ &- \frac{\delta t}{\tau_\sigma}\left[f_i^{\sigma,N}(\mathbf{r},t) - f_i^{\sigma,N,eq}(\mathbf{r},t)\right] \end{aligned}$$

$$\sigma = 1, 2 \qquad \text{and} \qquad i = 0, 1, \dots N \tag{2.6}$$

where τ_σ, $\sigma = 1, 2$ are physical relaxation times and $f_i^{\sigma,N,eq}(\mathbf{r},t)$ are the equilibrium distribution functions.

The local number densities are defined by

$$n_\sigma(\mathbf{r},t) = \sum_{i=0}^{N} f_i^{\sigma,N}(\mathbf{r},t) = \sum_{i=0}^{N} f_i^{\sigma,N,eq}(\mathbf{r},t) \tag{2.7}$$

while the local density and the local velocity of each component are

$$\rho_\sigma(\mathbf{r},t) = m_\sigma n_\sigma(\mathbf{r},t) \tag{2.8}$$

$$\mathbf{u}_\sigma(\mathbf{r},t) = \frac{1}{n_\sigma(\mathbf{r},t)} \sum_{i=1}^{N} \mathbf{e}_i^{\sigma,N} f_i^{\sigma,N}(\mathbf{r},t) \tag{2.9}$$

The mass averaged (barycentric) velocity is defined by

$$\mathbf{u} = \frac{m_1 n_1 \mathbf{u}_1 + m_2 n_2 \mathbf{u}_2}{m_1 n_1 + m_2 n_2} = \frac{\sum_{\sigma=1}^{\sigma=2} \rho_\sigma \mathbf{u}_\sigma}{\sum_{\sigma=1}^{\sigma=2} \rho_\sigma} \tag{2.10}$$

The mass flux $\mathbf{j}_\sigma$ of the component σ [12, 13] is defined with respect to the barycentric velocity

$$\mathbf{j}_\sigma = rho_\sigma (\mathbf{u}_\sigma - \mathbf{u} \tag{2.11}$$

As usual in LB models, the equilibrium density functions are expressed as power series in the components of the local equilibrium velocity, $\mathbf{u}^{\sigma,N,eq}$

$$f_i^{\sigma,N,eq} = w_i^N n_\sigma(\mathbf{r},t) \tag{2.12}$$

$$\times \left[1 + \frac{\mathbf{e}_i^{\sigma,N} \cdot \mathbf{u}^{\sigma,N,eq}}{\chi_N c_{\sigma,N}^2} + \frac{(\mathbf{e}_i^{\sigma,N} \cdot \mathbf{u}^{\sigma,N,eq})^2}{2\chi_N c_{\sigma,N}^4} - \frac{(\mathbf{u}^{\sigma,N,eq})^2}{2\chi_N c_{\sigma,N}^2}\right]$$

where the weight factors w_i^N are

$$w_i^6 = \begin{cases} \frac{1}{2} & , \quad i = 0 \\ \frac{1}{12} & , \quad i = 1, \ldots, 6 \end{cases} \tag{2.13}$$

for the seven bit model, and

$$w_i^8 = \begin{cases} \frac{4}{9} & , \quad i = 0 \\ \frac{1}{9} & , \quad i = 1, \ldots 4 \\ \frac{1}{36} & , \quad i = 5, \ldots, 8 \end{cases} \tag{2.14}$$

for the nine bit model.

Since the elements of the velocity sets $\{\mathbf{e}_i^{\sigma,N}\}$ satisfy

$$\sum_i e_{i\alpha}^{\sigma,N} = 0$$

$$\sum_i w_i^N e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} = \chi_\sigma \, c_{\sigma,N}^2 \, \delta_{\alpha\beta} \tag{2.15}$$

$$\sum_i w_i^N e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} e_{i\gamma}^{\sigma,N} = 0$$

$$\sum_i w_i^{\sigma,N} e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} e_{i\gamma}^{\sigma,N} e_{i\delta}^{\sigma,N} = \chi_\sigma^2 \, c_{\sigma,N}^4 \, (\delta_{\alpha\beta}\delta_{\gamma\delta} + \delta_{\gamma\beta}\delta_{\alpha\delta} + \delta_{\delta\beta}\delta_{\gamma\alpha})$$

following sums are easily computed

$$\sum_{i=0}^{i=\sigma} f_i^{\sigma,N,eq} = n_\sigma \tag{2.16}$$

$$\sum_{i=0}^{i=\sigma} e_{i\alpha}^{\sigma,N} f_i^{\sigma,N,eq} = n_\sigma(\mathbf{r},t) u_\alpha^{\sigma,N,eq} \tag{2.17}$$

$$\sum_{i=0}^{i=\sigma} e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma}, f_i^{\sigma,N,eq} = n_\sigma \left[\chi_\sigma \, c_\sigma^2 \, \delta_{\alpha\beta} + u_\alpha u_\beta \right] \tag{2.18}$$

$$\sum_{i=0}^{i=\sigma} e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} e_{i\gamma}^{\sigma,N} f_i^{\sigma,N,eq} = \chi_\sigma \, c_{\sigma,N}^2 \, n_\sigma \, (\delta_{\alpha\beta} u_\gamma + \delta_{\alpha\gamma} u_\beta + \delta_{\beta\gamma} u_\alpha) \tag{2.19}$$

As discussed previously [1], the macroscopic behavior of a single component fluid system is independent of the discretization of the phase space (i.e., independent of the number of elements N of the discrete velocity set $\{\mathbf{e}_i^{\sigma,N}\}$ used in the FDLB model). In the case of a two component system, we will consider the same model (seven bit ot nine bit, i.e., the same N) for both components. The choice $N = 6$ or $N = 8$ is only a technical detail related to the FDLB computer code and has no influence on the results at the macroscopic scale. To reduce the number of indices in the following formulae in this chapter (as well as in the subsequent chapters of this report), we will discard the index N. Consequently, we will write

$$\partial_t f_i^\sigma(\mathbf{r},t) + e_{i\alpha}^\sigma \partial_\alpha f_i^\sigma(\mathbf{r},t) = -\frac{1}{\tau_\sigma} \left[f_i^\sigma(\mathbf{r},t) - f_i^{\sigma,eq}(\mathbf{r},t) \right]$$

$$\sigma = 1, 2 \qquad \text{and} \qquad i = 0, 1, \ldots N \tag{2.20}$$

instead of Eq. (2.5),

$$f_i^{\sigma,eq} = w_i \, n_\sigma(\mathbf{r},t) \left[1 + \frac{\mathbf{e}_i^\sigma \cdot \mathbf{u}^{\sigma,eq}}{\chi_\sigma c_\sigma^2} + \frac{(\mathbf{e}_i^\sigma \cdot \mathbf{u}^{\sigma,eq})^2}{2\chi_\sigma^2 c_\sigma^4} - \frac{(\mathbf{u}^{\sigma,eq})^2}{2\chi_\sigma c_\sigma^2} \right] \tag{2.21}$$

instead of Eq. (2.13), and so on.

2.2 Chapman - Enskog expansion

The distribution functions are usually expanded as power series in the Knudsen number ε

$$f_i^\sigma = f_i^{\sigma(0)} + \varepsilon f_i^{\sigma(1)} + \varepsilon^2 f_i^{\sigma(2)} + \ldots \tag{2.22}$$

while two corresponding time scales and one length scale are introduced when computing time and space derivatives

$$\partial_t \equiv \varepsilon \partial_{t_1} + \varepsilon^2 \partial_{t_2} \tag{2.23}$$

$$\partial_\alpha \equiv \varepsilon \partial_\alpha \tag{2.24}$$

After introducing the Chapman - Enskog expansion (2.22), as well as the corresponding expansions (2.23) and (2.24) of the time and space derivatives in the Boltzmann equation (2.20)

$$\begin{aligned}\left(\varepsilon \partial_{t_1} + \varepsilon^2 \partial_{t_2}\right)\left[f_i^{\sigma(0)} + \varepsilon f_i^{\sigma(1)} + \varepsilon^2 f_i^{\sigma(2)} + \ldots\right] &+ \\ \varepsilon e_{i\beta}^\sigma \partial_\beta \left[f_i^{\sigma(0)} + \varepsilon f_i^{\sigma(1)} + \varepsilon^2 f_i^{\sigma(2)} + \ldots\right] &= \\ -\frac{1}{\tau_\sigma}\left[f_i^{\sigma(0)} + \varepsilon f_i^{\sigma(1)} + \varepsilon^2 f_i^{\sigma(2)} + \ldots - f_i^{\sigma,eq}\right] &\end{aligned} \tag{2.25}$$

we can separate the zero-th, first and second order Boltzmann equations with respect to the Knudsen number ε

$$0 = -\frac{1}{\tau_\sigma}\left[f_i^{\sigma(0)} - f_i^{\sigma,eq}\right] \tag{2.26}$$

$$\partial_{t_1} f_i^{\sigma(0)} + \partial_\beta f_i^{\sigma(0)} e_{i\beta}^\sigma = -\frac{1}{\tau_\sigma} f_i^{\sigma(1)} \tag{2.27}$$

$$\partial_{t_2} f_i^{\sigma(0)} + \partial_{t_1} f_i^{\sigma(1)} + \partial_\beta f_i^{\sigma(1)} e_{i\beta}^\sigma = -\frac{1}{\tau_\sigma} f_i^{\sigma(2)} \tag{2.28}$$

From the zero-th order Boltzmann equation (2.26), we get, for any $\sigma \in \{1, 2\}$ and $i = 0, 1 \ldots N$

$$f_i^{\sigma(0)} = f_i^{\sigma,eq} \tag{2.29}$$

This condition ensures also the validity of the zero-th order mass and momentum equations (2.32) and (2.35), which will be discussed in the next section.

Taking into account the expression (2.7) of the local number density n_σ and the series expansion (2.22), we get

$$\sum_{i=0}^{i=N} f_i^{\sigma(l)} = 0 \qquad \forall l \geq 1 \tag{2.30}$$

As usual in many Lattice Boltzmann models for multicomponent fluids [13, 14, 15, 16, 17], we assume that all components have the same equilibrium velocity in the absence of external forces and long range interactions

$$u_\alpha^{\sigma,eq} \equiv u'_\alpha \qquad \forall\sigma = 1,2 \qquad \text{and} \qquad \alpha = 1,2 \tag{2.31}$$

Consequently, the equilibrium distribution functions $f_i^{\sigma,eq} \equiv f_i^{\sigma(0)}$ are expressed as series expansions (2.21) with respect to the components u'_α of this equilibrium velocity. At this stage, we are not interested in the procedure which should provide the means to compute the values of the components u'_α; this procedure will be discussed later.

2.3 Conservation equations

The zero-th, first and second order mass conservation equations for each component $\sigma = 1, 2$ are recovered from Eqs. (2.26 – 2.28) after summation with respect to i (multiplication by m_σ is omitted)

$$-\frac{1}{\tau_\sigma} \sum_{i=0}^{i=N} \left[f_i^{\sigma(0)} - f_i^{\sigma,eq} \right] = 0 \tag{2.32}$$

$$\partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_\beta \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\beta}^\sigma = -\frac{1}{\tau_\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(1)} \tag{2.33}$$

$$\partial_{t_2} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(1)} + \partial_\beta \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\beta}^\sigma = -\frac{1}{\tau_\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(2)} \tag{2.34}$$

The zero-th, first and second order momentum conservation equations are also recovered from Eqs. (2.26 – 2.28) after multiplication with $m_\sigma e_{i\alpha}^\sigma$ and summation with respect to i and σ

$$-\sum_{\sigma=1}^{\sigma=2} \frac{m_\sigma}{\tau_\sigma} \sum_{i=0}^{i=N} \left[f_i^{\sigma(0)} - f_i^{\sigma,eq} \right] e_{i\alpha}^\sigma = 0 \tag{2.35}$$

$$\partial_{t_1} \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma + \partial_\beta \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma e_{i\beta}^\sigma = - \sum_{\sigma=1}^{\sigma=2} \frac{m_\sigma}{\tau_\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\alpha}^\sigma \tag{2.36}$$

$$\partial_{t_2} \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma + \partial_{t_1} \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\alpha}^\sigma + \partial_\beta \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\alpha}^\sigma e_{i\beta}^\sigma = - \sum_{\sigma=1}^{\sigma=2} \frac{m_\sigma}{\tau_\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(2)} e_{i\alpha}^\sigma \tag{2.37}$$

The zero-th order mass and momentum equations (2.32) and (2.35) are automatically satisfied in accordance to Eq. (2.29). Moreover, the right hand sides of the first and second order mass equations (2.33) and (2.34) vanish because of Eq. (2.30) and thus, these equations are rewritten as

$$\partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_\beta \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\beta}^\sigma = 0 \tag{2.38}$$

$$\partial_{t_2} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(1)} + \partial_\beta \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\beta}^\sigma = 0 \tag{2.39}$$

2.4 Mass equations

Since $f_i^{\sigma(0)} = f_i^{\sigma,eq}$, we can use the series expansion (2.21), as well as the properties (2.16) and (2.17), to rewrite the first order mass conservation equation (2.38) in a more familiar form, after multiplication by m_σ

$$\partial_{t_1} \rho_\sigma + \partial_\beta (\rho_\sigma u'_\beta) = 0 \tag{2.40}$$

Here

$$\rho_\sigma = m_\sigma n_\sigma \tag{2.41}$$

is the local density of the component σ.

Using the first order Boltzmann equation (2.27) to express $f_i^{\sigma(1)}$ in the second order equation (2.28), we get

$$\partial_{t_2} f_i^{\sigma(0)} + \tau_\sigma (\partial_{t_1})^2 f_i^{\sigma(0)} - 2\tau_\sigma \partial_{t_1} \left[\partial_{t_1} f^{\sigma(0)} + \partial_\beta f_i^{\sigma(0)} e_{i\beta}^\sigma \right] -$$

$$\tau_\sigma \partial_\alpha \partial_\beta f_i^{\sigma(0)} e_{i\alpha}^\sigma e_{i\beta}^\sigma = \frac{1}{\tau_\sigma} f_i^{\sigma(2)} \tag{2.42}$$

This equation, multiplied by m_σ and summed over i, gives the following form of the the second order mass conservation equation (2.39)

$$\partial_{t_2} \rho_\sigma + \tau_\sigma (\partial_{t_1})^2 \rho_\sigma - \tau_\sigma \partial_\alpha \partial_\beta (n_\sigma k_B T \delta_{\alpha\beta} + \rho_\sigma u'_\alpha u'_\beta) = 0 \tag{2.43}$$

To derive this result, we used the first order mass equation (2.40), as well as the property (2.18) and the definition (2.3).

To recover the mass conservation equation for the component σ up to the second order with respect to the Knudsen number, we sum together Eqs. (2.40) and (2.43) multiplied by ε and ε^2, respectively, and take into account the expressions (2.23) and (2.24) of the time and space derivatives

$$\begin{aligned} \partial_t \rho_\sigma + \partial_\alpha (\rho_\sigma u'_\alpha) + \tau_\sigma (\partial_t)^2 \rho_\sigma & - \\ \tau_\sigma \frac{k_B T}{m_\sigma} \delta_{\alpha\beta} \partial_\alpha \partial_\beta \rho_\sigma - \tau_\sigma \partial_\alpha \partial_\beta (\rho_\sigma u'_\alpha u'_\beta) & = 0 \end{aligned} \tag{2.44}$$

When summing over σ, we get also the mass equation for the whole fluid system

$$\begin{aligned} \partial_t \rho + \partial_\alpha (\rho u'_\alpha) + (\partial_t)^2 \sum_{\sigma=1}^{\sigma=2} \tau_\sigma \rho_\sigma & - \\ k_B T \, \partial_\alpha \partial_\beta \sum_{\sigma=1}^{\sigma=2} \frac{\tau_\sigma \rho_\sigma}{m_\sigma} - \partial_\alpha \partial_\beta \left[\left(\sum_{i=1}^{i=2} \tau_\sigma \rho_\sigma \right) u_\alpha u_\beta \right] & = 0 \end{aligned} \tag{2.45}$$

where

$$\rho = \sum_{\sigma=1}^{\sigma=2} \rho_\sigma \tag{2.46}$$

is the local density of the whole fluid. In the particular case $\tau_\sigma = \tau \; \forall \sigma$, the mass equation for the whole fluid (2.45) becomes

$$\partial_t \rho + \partial_\alpha (\rho u'_\alpha) + \tau (\partial_t)^2 \rho_\sigma - \tau \, \partial_\alpha \partial_\beta \, p - \tau \partial_\alpha \partial_\beta \left(\rho u'_\alpha u'_\beta \right) = 0 \tag{2.47}$$

where

$$p = k_B T \sum_{\sigma=1}^{\sigma=N} n_\sigma \tag{2.48}$$

is the local pressure of the fluid system, which is a mixture of two ideal gases.

The mass equation for a single component (2.44), as well as the mass equation for the whole system (2.45) contain the first and second order time derivatives of the local density. The presence of both time derivatives is a characteristics of the telegraphist equation [18], which describes a propagation phenomenon. Consequently, one may expect that the density profile across the system exhibits a kink (wrinkle or peak), i.e., a non monotonic propagating pulse. This behavior should be dominant for small values of the product between the local density and the equilibrium velocity ($\rho_\sigma u'_\alpha \simeq 0$), when the mass equation for the component σ reduces to the true telegraphist equation

$$\partial_t \rho_\sigma + \frac{\mathcal{D}}{\mathcal{C}} (\partial_t)^2 \rho_\sigma = \mathcal{D}\, \nabla^2 \rho_\sigma \tag{2.49}$$

where the diffusion coefficient is

$$\mathcal{D} = \tau_\sigma \frac{k_B T}{m_\sigma} \tag{2.50}$$

and

$$\mathcal{C} = \frac{k_B T}{m_\sigma} \tag{2.51}$$

is the finite speed at which information travels in the system. The diffusion equation is recovered in the case

$$\frac{\mathcal{D}}{\mathcal{C}} = \tau_\sigma \to 0 \tag{2.52}$$

2.5 Equilibrium velocity u′

The right hand side of the Boltzmann equation (2.20) represents the collision term which expresses the variation of the equilibrium distribution function as a result of collisions between particles. Collisions should preserve the local momentum of the whole system. Since the momentum equation is derived from the Boltzmann equation after multiplication by $m_\sigma e^\sigma_{i\alpha}$ and summation over the indices σ and i, we should have

$$\sum_{\sigma=1}^{\sigma=2} \frac{m_\sigma}{\tau_\sigma} \sum_{i=1}^{i=N} \left[f_i^\sigma - f_i^{\sigma,eq} \right] e^\sigma_{i\alpha} = 0 \tag{2.53}$$

Since $f_i^{\sigma,eq}$ is expressed as a series expansion (2.21 in the equilibrium velocity u'_α (which is supposed to be the same for all σ), we can use Eq. (2.17) to get

$$u'_\alpha \sum_{\sigma=1}^{\sigma=2} \frac{m_\sigma n_\sigma}{\tau_\sigma} = \sum_{\sigma=1}^{\sigma=2} \frac{m_\sigma}{\tau_\sigma} \sum_{i=1}^{i=N} f_i^\sigma e_{i\alpha}^\sigma \tag{2.54}$$

which allows the determination of the components u'_α of the equilibrium velocity in the node $\mathbf{r}$ of the lattice.

If we take into account the fact that the distribution functions f_i^σ are expressed as power series (2.22) in the Knudsen number, as well as the fact that $f_i^{\sigma(0)} = f_i^{\sigma,eq}$, we get the following relations from Eq. (2.54)

$$\sum_{\sigma=1}^{\sigma=2} \frac{m_\sigma}{\tau_\sigma} \sum_{i=1}^{i=N} f_i^{\sigma(l)} e_{i\alpha}^\sigma = 0 \qquad , \forall l \geq 1 \tag{2.55}$$

2.6 Momentum equation

If we use Eq. (2.55), the first order momentum equation (2.36) becomes

$$\partial_{t_1} \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma + \partial_\beta \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma e_{i\beta}^\sigma = 0 \tag{2.56}$$

Introducing the relations (2.17) and (2.18) in the above equation, we get the Euler equation

$$\partial_{t_1} \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha + \partial_\alpha p + \partial_\beta \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha u'_\beta = 0 \tag{2.57}$$

Using the first order Boltzmann equation (2.27) to express $f_i^{\sigma(1)}$ in the second order momentum equation (2.37), we get

$$\begin{aligned}
\partial_{t_2} \sum_{\sigma=1}^{\sigma=2} m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma + (\partial_{t_1})^2 \sum_{\sigma=1}^{\sigma=2} \tau_\sigma m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma & - \\
2\partial_{t_1} \left[\partial_{t_1} \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha + \partial_\alpha p + \partial_\beta \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha u'_\beta \right] & - \\
\partial_\beta \partial_\gamma \sum_{i=1}^{i=2} \tau_\sigma m_\sigma \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^\sigma e_{i\beta}^\sigma e_{i\gamma}^\sigma & = 0
\end{aligned} \tag{2.58}$$

The square bracket in the equation above vanishes in accordance to Eq. (2.57). If we take into account Eqs. (2.3) and (2.19) and consider $\tau_1 = \tau_2 = \tau$, the second order momentum equation becomes

$$\partial_{t_2} \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha \; + \; \tau \left(\partial_{t_1}\right)^2 \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha \\ - \; \tau k_B T \sum_{\sigma=1}^{\sigma=2} \left(2\partial_\alpha \nabla \cdot \mathbf{u}' + \nabla^2 u'_\alpha\right) = 0 \qquad (2.59)$$

When adding the first and second ordet momentum equations, we get

$$\partial_t \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha \; + \; \tau \left(\partial_t\right)^2 \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha + \partial_\alpha p + \partial_\beta \sum_{\sigma=1}^{\sigma=2} \rho_\sigma u'_\alpha u'_\beta \\ - \; \tau k_B T \sum_{\sigma=1}^{\sigma=2} \left[2\partial_\alpha (\nabla \cdot \mathbf{u}') + \nabla^2 u'_\alpha\right] = 0 \qquad (2.60)$$

Thus, the momentum equation up to the second order contains the first and second order time derivatives, like the mass equation of the whole fluid system (2.45. As mentioned previously, the presence of the second order time derivative is a characteristics of a propagation phenomenon [18].

2.7 Pure diffusion

We suppose that the fluid system is always at rest. In this case, the equilibrium velocity u'_α should vanish everywhere. When

$$u'_\alpha = 0 \qquad (2.61)$$

the series expansion (2.21) reduces to

$$f_i^{\sigma(0)} = f_i^{\sigma,eq} = w_i n_\sigma \qquad (2.62)$$

Consequently, the first order mass equations (2.38) becomes

$$\partial_{t_1} \sum_{i=0}^{i=N} w_i n_\sigma + \partial_\beta \sum_{i=0}^{i=N} w_i n_\sigma e_{i\beta}^\sigma = 0 \qquad (2.63)$$

In accordance to Eqs. (2.13) and (2.14), we have the following equalities

$$\sum_{i=0}^{i=N} w_i = 1 \tag{2.64}$$

$$\sum_{i=0}^{i=N} w_i e_{i\beta}^{\sigma} = 0 \tag{2.65}$$

which are valid for $N = 6$, as well as for $N = 8$ (i.e., independent of the number of the discrete velocity speeds in the phase space). Thus, the first order mass equation (2.63) reduces to

$$\partial_{t_1} n_\sigma = 0 \tag{2.66}$$

Therefore, from the first order Boltzmann equation (2.27) we get, using the expression (2.62)

$$f_i^{\sigma(1)} = -\tau_\sigma w_i e_{i\alpha}^{\sigma} \partial_\alpha n_\sigma \tag{2.67}$$

which may be introduced in the second order mass equation (2.39) to give

$$\partial_{t_2} \sum_{i=0}^{i=N} w_i n_\sigma - \partial_{t_1} \sum_{i=0}^{i=N} \tau_\sigma w_i e_{i\alpha}^{\sigma} \partial_\alpha n_\sigma - \partial_\beta \sum_{i=0}^{i=N} \tau_\sigma w_i e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} \partial_\alpha n_\sigma = 0 \tag{2.68}$$

Since

$$\sum_{i=0}^{i=N} w_i e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} = \chi_\sigma c_\sigma^2 \delta \alpha\beta \tag{2.69}$$

the second order equation (2.68) is rewritten as

$$\partial_{t_2} n_\sigma = \tau_\sigma \chi_\sigma c_\sigma^2 \nabla n_\sigma \tag{2.70}$$

After multiplication of the mass equations (2.63) and (2.70) by ε and ε^2, respectively, we can sum them together to recover the mass equation up to second order with respect to the Knudsen number

$$\partial_t n_\sigma = \tau_\sigma \chi_\sigma c_\sigma^2 \nabla n_\sigma \tag{2.71}$$

which, after multiplication with m_σ, becomes identical to the pure diffusion equation

$$\partial_t \rho_\sigma = \mathcal{D} \nabla^2 \rho_\sigma \tag{2.72}$$

where the diffusion coefficient is

$$\mathcal{D} = \tau_\sigma \chi_\sigma c_\sigma^2 = \tau_\sigma \frac{k_B T}{m_\sigma} \tag{2.73}$$

If we remind the expression of the viscosity in the FDLB model [1]

$$\nu = \tau_\sigma \chi_\sigma c_\sigma^2 = \tau_\sigma \frac{k_B T}{m_\sigma} \tag{2.74}$$

we see that the Schmidt number Sc [12] is a constant equal to unity in the present FDLB model

$$Sc = \frac{\nu}{\mathcal{D}} = 1 \tag{2.75}$$

2.8 Comparison with the theory of Shan and Doolen

The LGLB model previously developed by Shan and Doolen [13, 17] was used to simulate the time evolution of the barycentric velocity profile in a diffusion couple [19]. Here is a brief outline of the main characteristics of this model:

1. All components have the same equilibrium velocity $\mathbf{u}'$, which means that the hypothesis expressed by Eq. (2.31) is used.

2. The equilibrium velocity $\mathbf{u}'$ in the absence of external forces is given by Eq (2.54).

3. The equilibrium distribution functions $f_i^{\sigma,eq}$ are expressed as series expansions in the equilibrium velocity $\mathbf{u}'$ while the leading order distribution functions $f_i^{\sigma(0)}$ are expressed as series expansions in the local fluid velocity $\mathbf{u}$, (i.e., the barycentric velocity – see [17], as well as [19]); this means that Eq. (2.29) is not valid in the model of Shan and Doolen; in fact, they make no explicit use of the zero-th, first and second order Boltzmann equations (2.26 – 2.28) as separate entities; we may imagine that the starting point of the model of Shan and Doolen is the sum of the zero-th and first order Boltzmann equation

$$\partial_{t_1} f_i^{\sigma(0)} + \partial_\beta f_i^{\sigma(0)} e_{i\beta}^\sigma = -\frac{1}{\tau_\sigma}\left[f_i^{\sigma(0)} + f_i^{\sigma(1)} - f_i^{\sigma,eq}\right] \tag{2.76}$$

which allows $f_i^{\sigma(0)} \neq f_i^{\sigma,eq}$, and so on [19].

4. The series expansion for $f_i^{\sigma,eq}$ and $f_i^{\sigma(0)}$ which is valid for $N = 6$, i.e., the six bit model (see [19]) contains the parameter d_σ which may be adjusted to allow different values of the diffusivity; this "degree of freedom" does not exist in the FDLB model since the very recent approach of He et al. [20, 21, 22] clearly establishes the value $d_\sigma = 0.5$ using Gaussian quadrature formulae.

5. We mention here that Shan and Doolen give two simulation results (figures 1 and 2 in their paper [13]); these figures refer to equilibrium density profiles in a binary mixture which seem to be correct; since these results refer to systems at equilibrium, we think that the barycentric velocity $\mathbf{u}$, as well as the equilibrium velocity $\mathbf{u}'$ vanish and the pure diffusion case is recovered; this may be a serious argument to consider the model as being valid only for stationary (equilibrium) cases, when there is some competition between diffusion and other phenomena generated by external forces, while the model itself does not account for the true dynamic evolution towards the equilibrium (stationary) case.

6. We should mention also the fact that the LGLB model of Shan and Doolen inherits the main disadvantage of the LGLB models, which is the fact that the particle thermal speeds in this model are strictly related to the lattice spacing (in fact, the magnitude of these speeds is strictly the lattice spacing divided by the time step) and thus, any LGLB model does not allow different thermal speed for particles carrying different masses; FDLB models allow different thermal speeds when the particles have differents masses and thus, one may expect these models to be more close to the physical reality.

2.9 Simulation results

We used a 250×5 square lattice with walls placed left and right. The lattice spacing was $\delta x = \delta y = 10^{-4}$ m. Periodic conditions were used at the upper and bottom boundaries. The left half of the lattice was initialized with particles belonging to species $\sigma = 1$ while the right half was initialized with particles belonging to species $\sigma = 2$. When developing the computer code, we used the nine bit model ($N = 8$) since this model allowed a larger variety of finite differences schemes to be tested (see Chapter 1).

Our first diffusion code used the Second Order Runge Kutta for time integration [1], combined with an Upwind scheme for calculating space derivatives. Later, we developed a code based on the First Order Upwind Scheme (FU) which we described in the previous chapter. Both the Runge Kutta code (Appendix B), as well as the FU code (Appendix A) gave similar results which are described below.

The initial particle number density was set to unity for each component. As a result, the initial density profiles of each component, as well as the total density profile were always similar to those depicted in Figure 2.1, where we used $m_1 = 1$ amu and $m_2 = 0.9$ amu. 1 amu (atomic mass unit) equals 1.661×10^{-27} kg.

The present simulations were done with the same value of the relaxation time: $\tau_\sigma = \tau = 10^{-8}$ s, while the time step was chosen to be $\delta t = 10^{-9}$ s. The thermal velocities of each species of particles were calculated in accordance to Eq. (2.3) with $T = 300$ K.

Figure 2.2 shows the typical time evolution of the total density profile. One can see the presence of two kinks: a left propagating kink and a right propagating one. Their evolution is presented in Figures 2.3 and 2.4. The presence of these propagating kinks is not a surprise if we remind the mass equation (2.45) of the whole fluid system, which contains the second order time derivative. As mentioned before, the presence of the second order time derivative is a characteristics of a propagation phenomena.

Figure 2.5 shows the time evolution of the barycentric velocity. We may see the two kinks which form and propagate laterally, while the profile exhibits a central peak which remains always positive. The propagating kinks are reflected by the lateral walls and superpose to the central peak at some moments during their propagation. The formation of these propagating kinks in the figures showing the time evolution of the barycentric velocity may be explained by the presence of the second order time derivative in the momentum equation (2.60).

Figure 2.6 shows the time evolution of the mass flux of component 2. This flux is orientated from left to right, as expected, and does not change its sign. No kinks are observed.

Figure 2.7 shows the total density profile when the two species of particles have the same mass. This profile is found to be constant during the diffusion process, while the individual profiles of components 1 and 2 evolve separately. Figure 2.8 shows the corresponding time evolution of the local density of component 1. One can see that kinks are not present in the den-

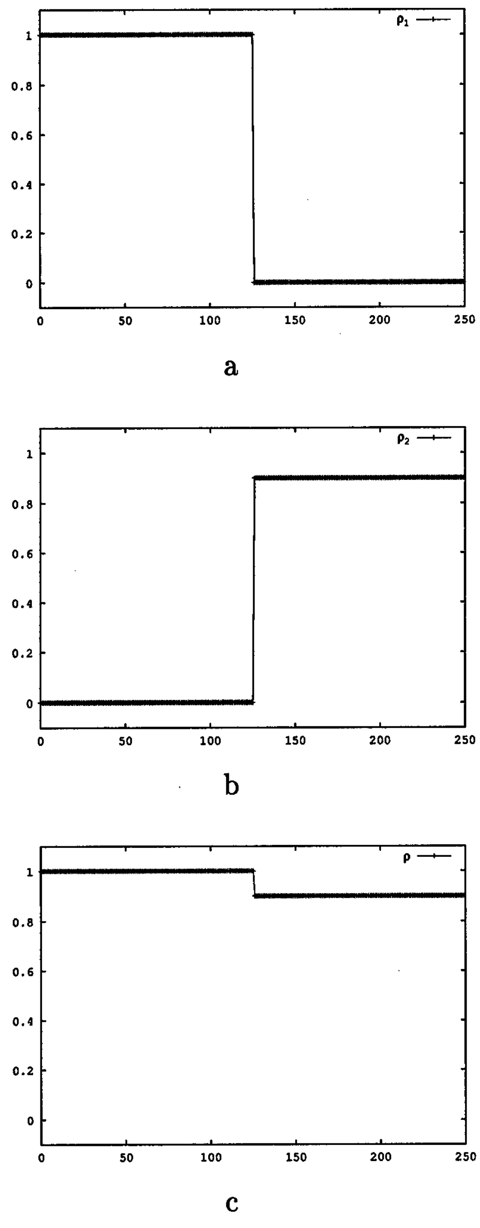
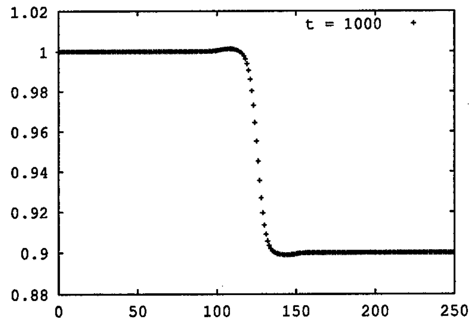
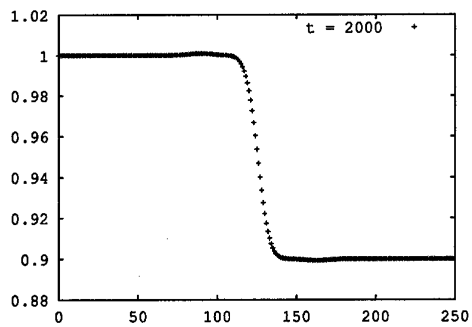


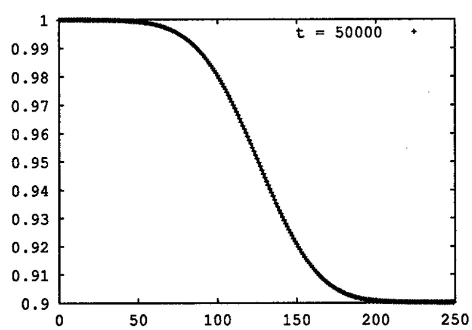
Figure 2.1: Initial density profiles in the diffusion couple: a – component $\sigma = 1$; b – component $\sigma = 2$; c – total density profile $\rho = \rho_1 + \rho_2$.



a: 1,000 time steps



b: 2,000 time steps



c: 50,000 time steps

Figure 2.2: Time evolution of the total density profiles in the diffusion couple.

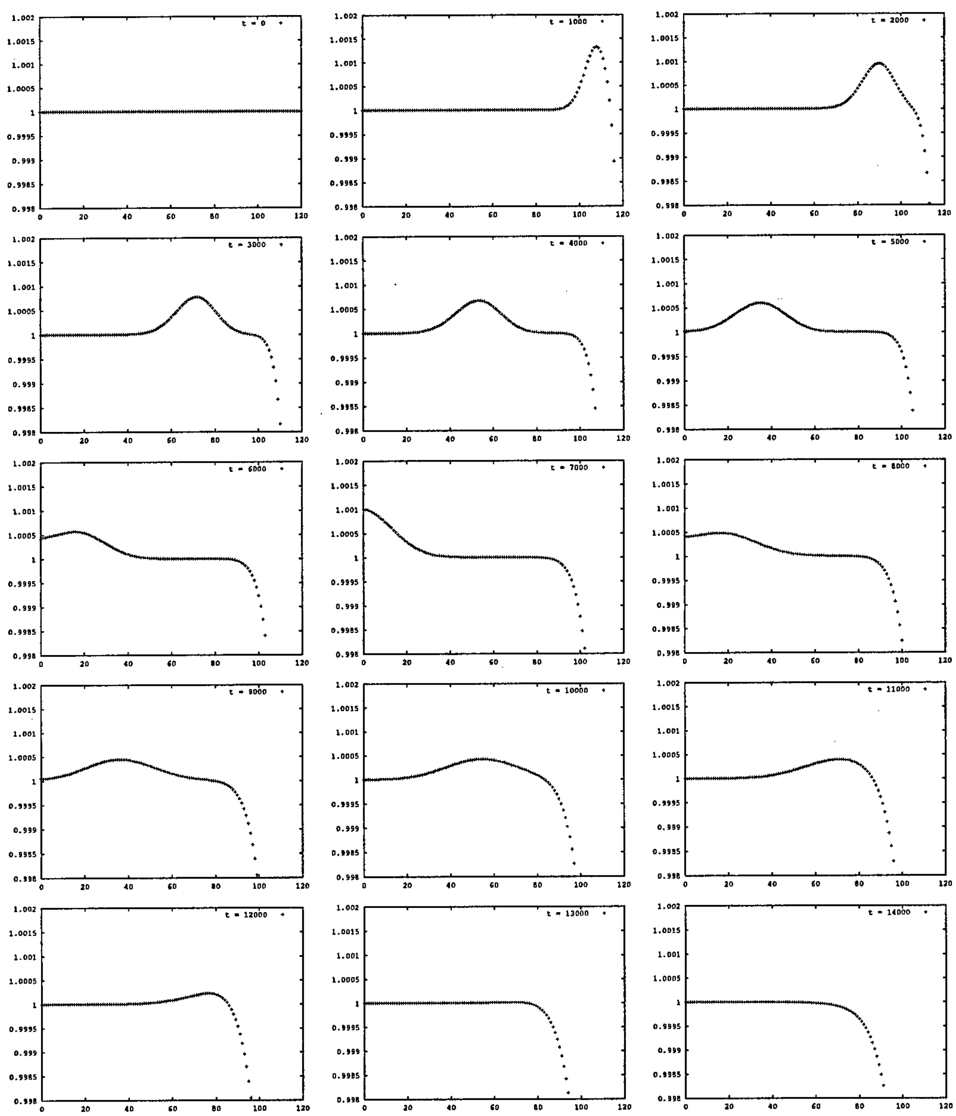


Figure 2.3: Time evolution of the total density profiles in the left region of the diffusion couple (snapshots are taken every 1000 time steps).

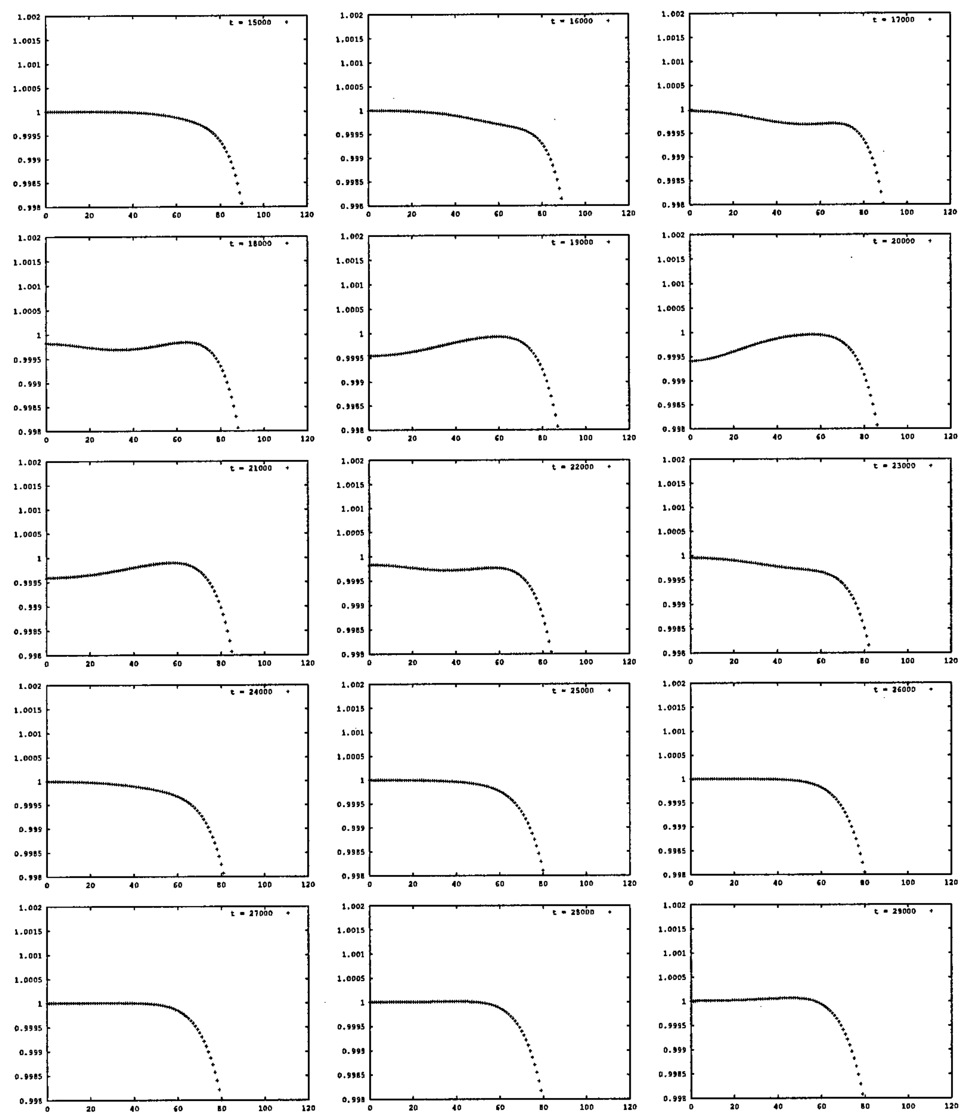


Figure 2.3: (continued) Time evolution of the total density profiles in the left region of the diffusion couple (snapshots are taken every 1000 time steps).

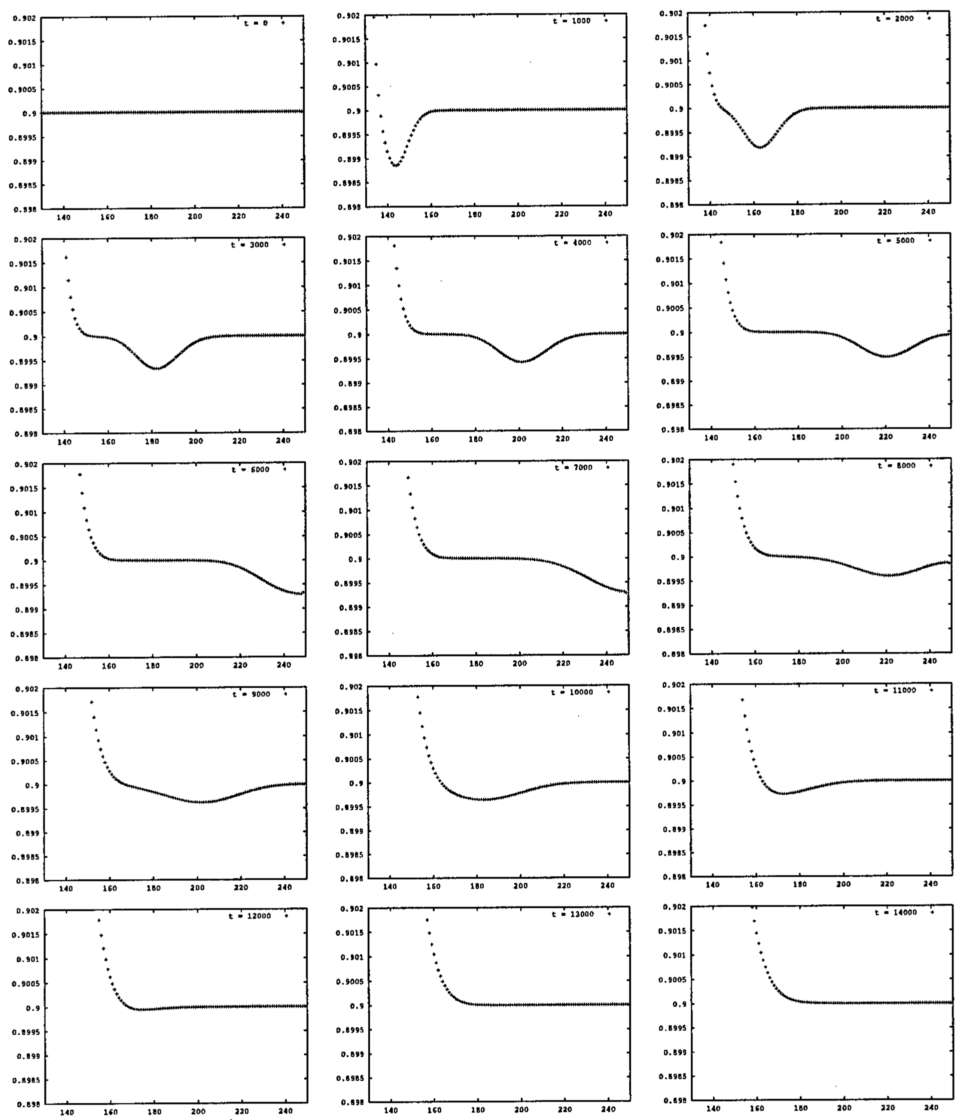


Figure 2.4: Time evolution of the total density profiles in the right region of the diffusion couple (snapshots are taken every 1000 time steps).

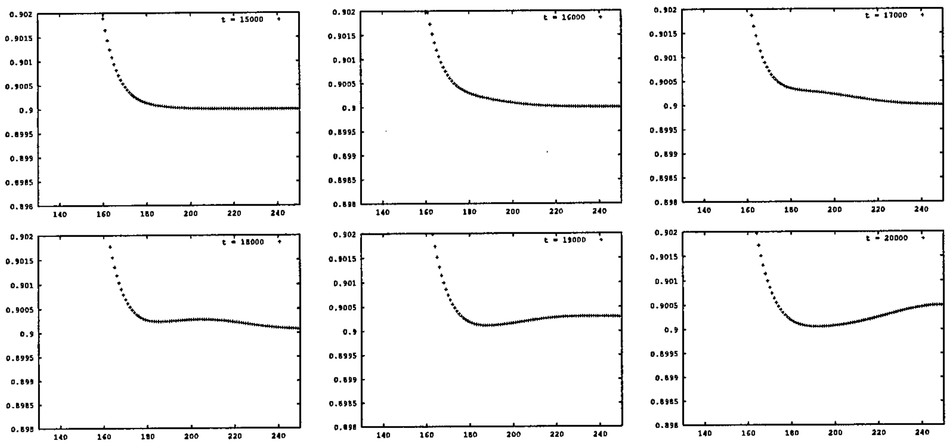


Figure 2.4: (continued) Time evolution of the total density profiles in the right region of the diffusion couple (snapshots are taken every 1000 time steps).

sity profiles when masses are equals, which means that we are dealing with a pure diffusion process.

To compare the FDLB simulations wit the former LGLB simulations [19], we slightly modified the corresponding computer code and set the same thermal velocity for each species of particles, even if the masses are different. For convenience, we used $m_1 = 1$ amu, $m_2 = 0.9$ amu, while c_1 was given by Eq. (2.3) and c_2 was forced to be equal to c_1. Figure 2.9 shows the time evolution of the barycentric velocity in the diffusion couple, which is very similar to the LGLB results reported in [19]. In this case, the barycentric velocity changes its sign alternatively, due to strong oscillations which are present in the system.

The unphysical behavior of the barycentric velocity reported in figure 2.9, as well as the simulation results reported in [19] provide a very strong reason for a careful handling of the former LGLB models when dealing with multicomponent (e.g., binary) systems whose particles carry different masses. All the present Lattice Boltzmann litterature dealing with multicomponent systems, e.g., the approaches in [11, 14, 18, 23, 24, 25, 26, 27, 28, 29, 30], ignores completely the connection between the lattice spacing and the thermal speeds of particles and do not report unphysical effects of the Lattice Gas like

Lattice Boltzmann scheme. The unphysical oscillations which are observed when trying to simulate the behavior of a diffusion couple in microgravity environment with the LGLB model revealed a severe limitation of these models to those multicomponent systems whose particles have identical mass (as well as the same thermal velocity). Finite Difference Lattice Boltzmann Models (FDLB) or Interpolation Supplemented Lattice Boltzmann Models (ISLB) should be used in order to allow different thermal velocities of particles in the system (i.e., different values of the Courant - Friedrichs - Lewy number CFL). In this respect, the absence of the oscillations of the barycentric velocity in the diffusion couple, as reported in Figure 2.5, are very encouraging.

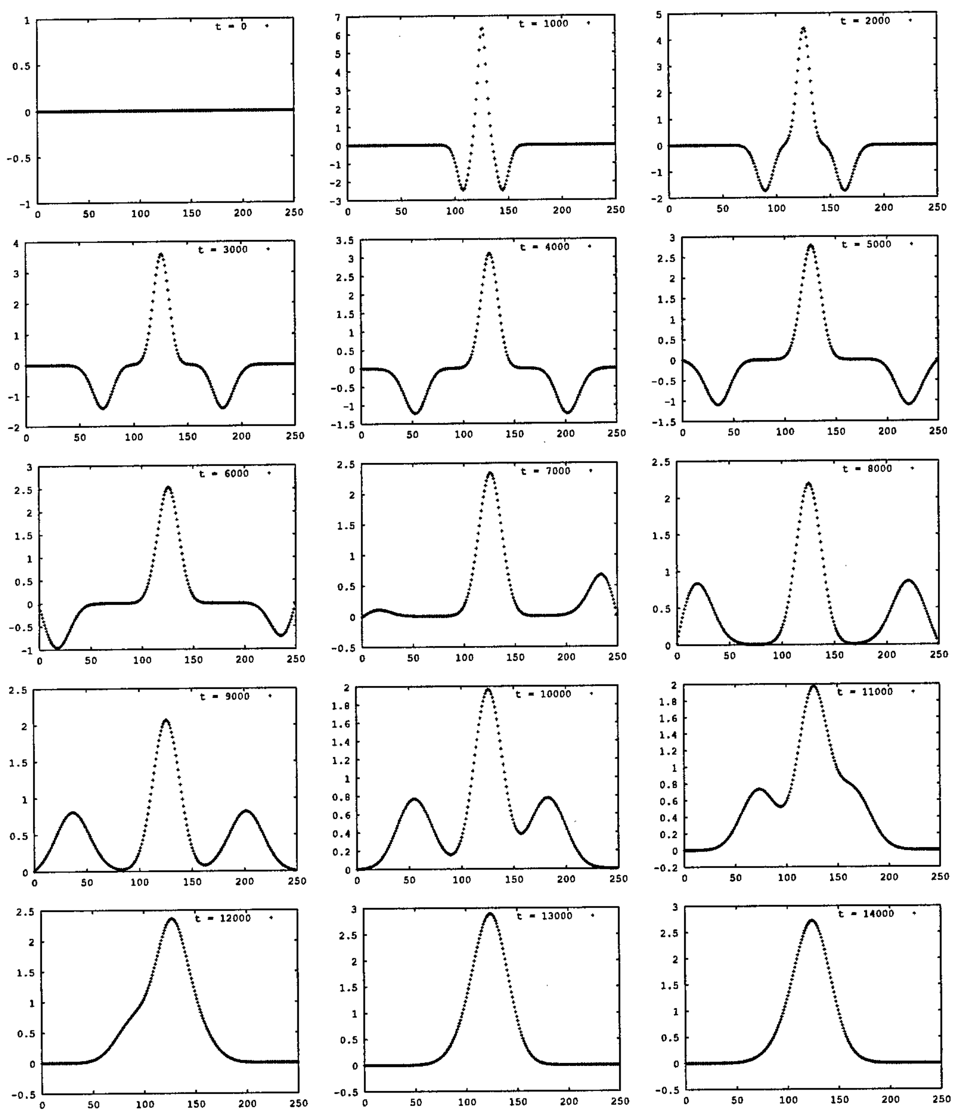


Figure 2.5: Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

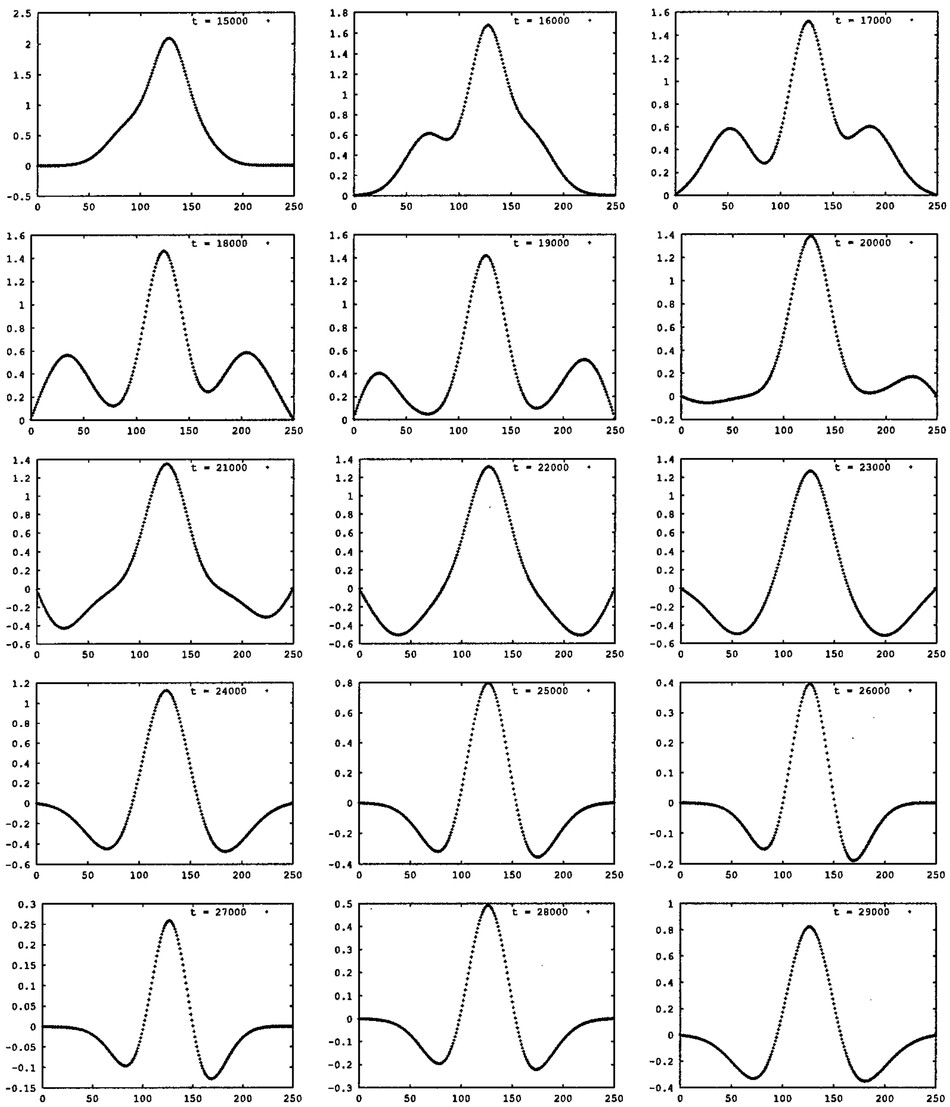


Figure 2.5: (continued) Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

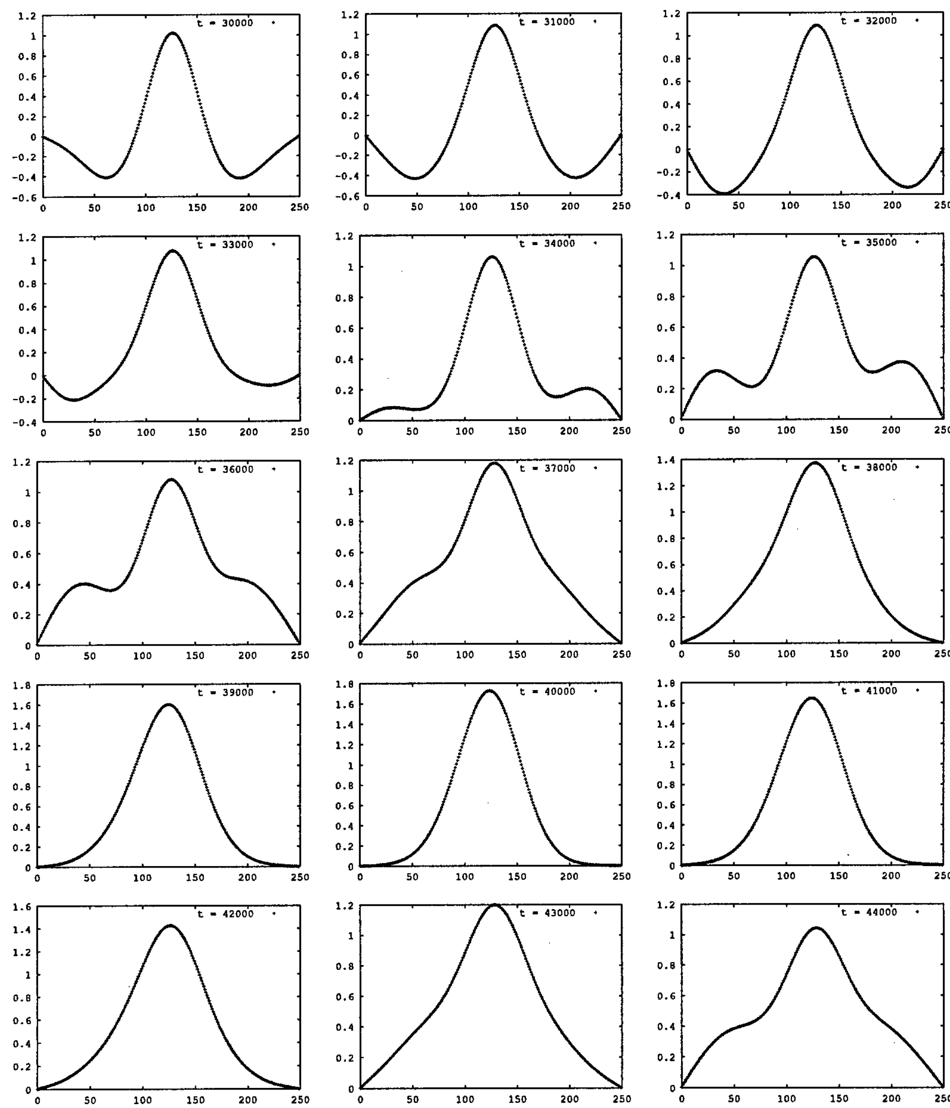


Figure 2.5: (continued) Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

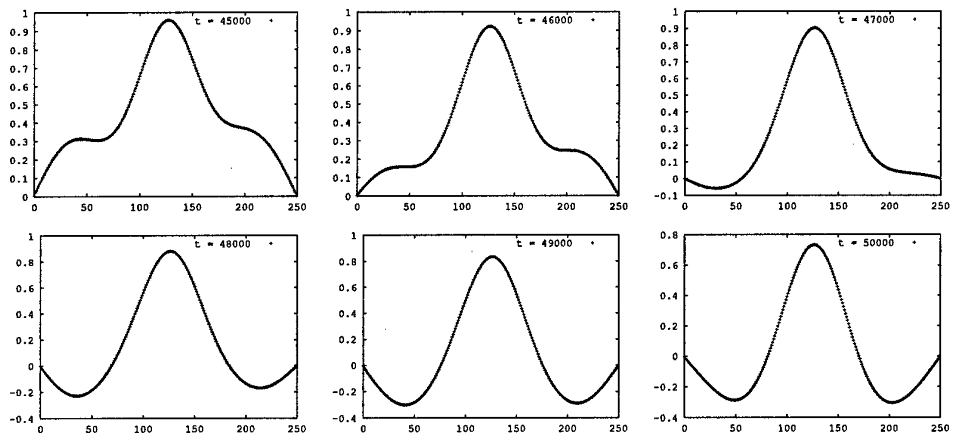


Figure 2.5: (continued) Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

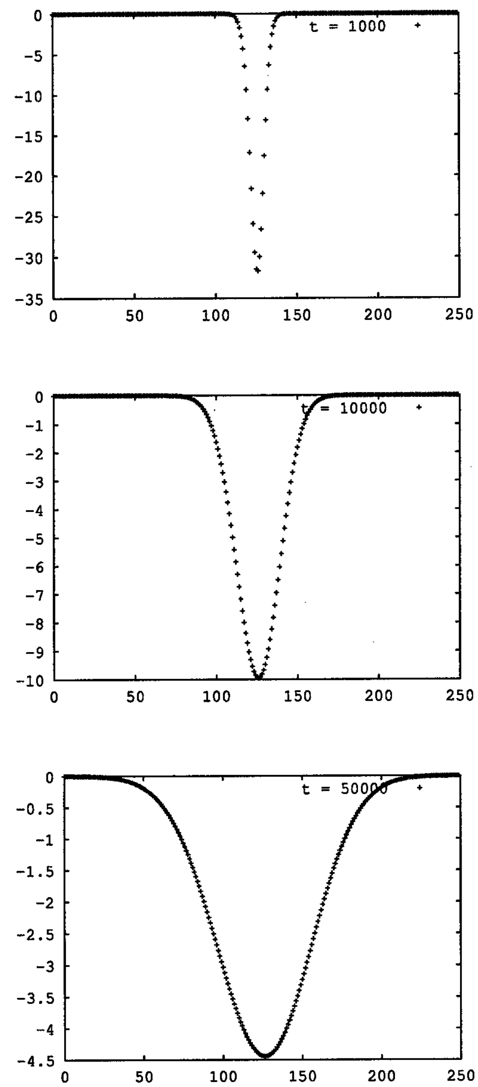


Figure 2.6: Time evolution of mass flux of component 2.

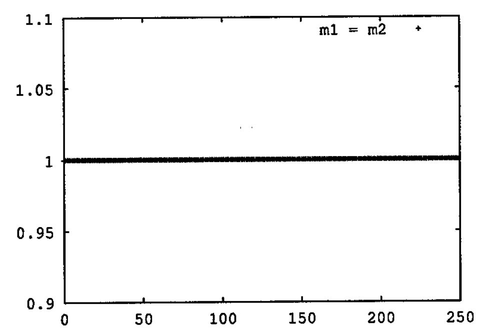


Figure 2.7: Total density profiles in the diffusion couple when the two species of particles have equall mass: this profile remains constant during the diffusion process.

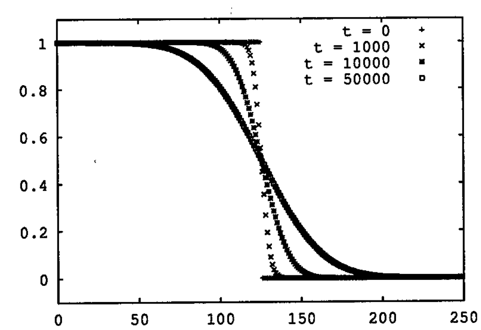


Figure 2.8: Time evolution of the density profiles of component 1 when the two species of particles have equall mass: no kinks are observed.

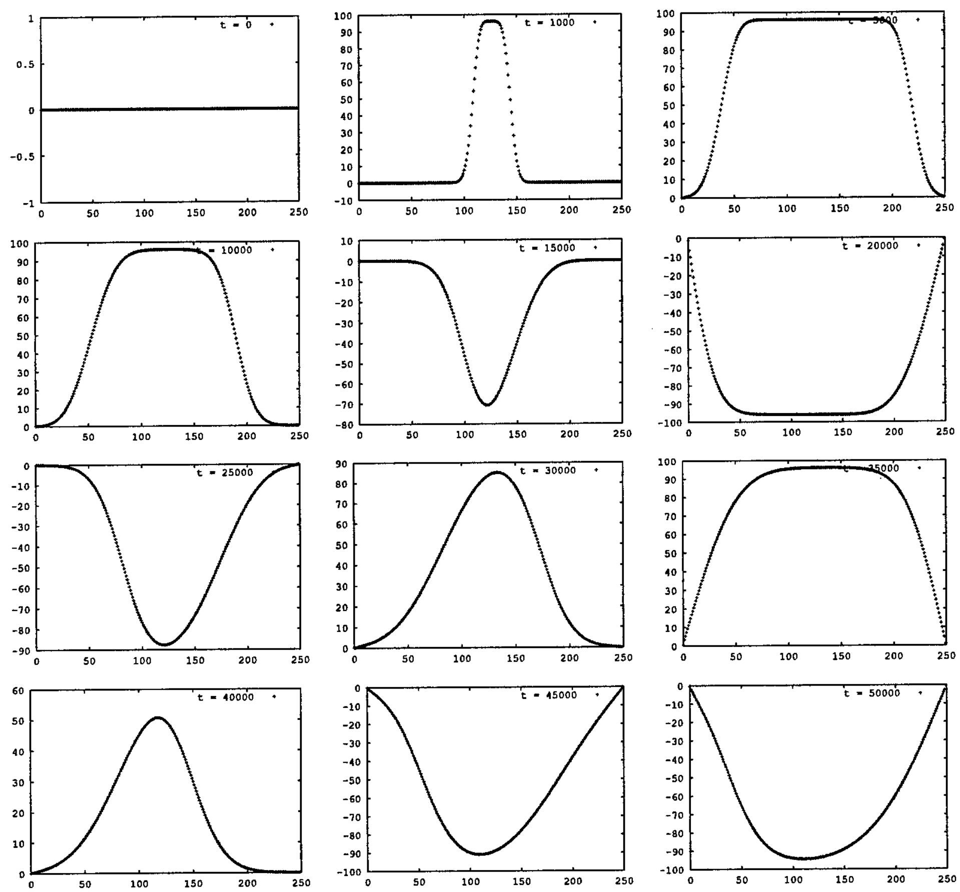


Figure 2.9: Time evolution of the barycentric velocity profile (snapshots are taken every 5000 time steps, except the second one, which is taken after 1000 time steps) when the two thermal velocities are set equal ($c_1 = c_2$).

Chapter 3

Sessile drop

3.1 Drop shape

A computer code based on a lattice Boltzmann model for liquid - vapor systems was already described in the preliminary report [31]. This code was used to investigate the equilibrium shape of a liquid drop on a horizontal surface and some preliminary results were presented in [31]. These preliminary simulations were done on a 256 $\times$ 64 lattice because of limitations in the computer resources available at that time. In the present report we will refer to subsequent simulations made on larger lattices, which were done on a Pentium II processor with 256 MB RAM working under the UNIX FreeBSD operating system. The code is given in Appendix C.

Following constant values of the main simulation parameters were used, as in [31]

- mean system density $\rho_m = 3.5$
- conventional temperature $T = 0.550$
- relaxation time $\tau = 0.8$

To investigate the shape of a sessile drop on a horizontal surface subjected to a gravitational field, we used a constant value of the parameter $\kappa = 0.01$ of the LB model, which defines the surface tension, as discussed in [31]. We considered the two different cases, when the liquid is wetting the horizontal surface or not. For the first case, we adopted the value

$$\frac{\partial \mu}{\partial x_n} = 0.01 \tag{3.1}$$

of the normal derivative of the chemical potential on the lower wall surface. This value of the derivative accounts for the wetting interaction between the liquid drop and the wall. The second case is characterized by

$$\frac{\partial \mu}{\partial x_n} = -0.01 \tag{3.2}$$

which accounts for a non - wetting wall.

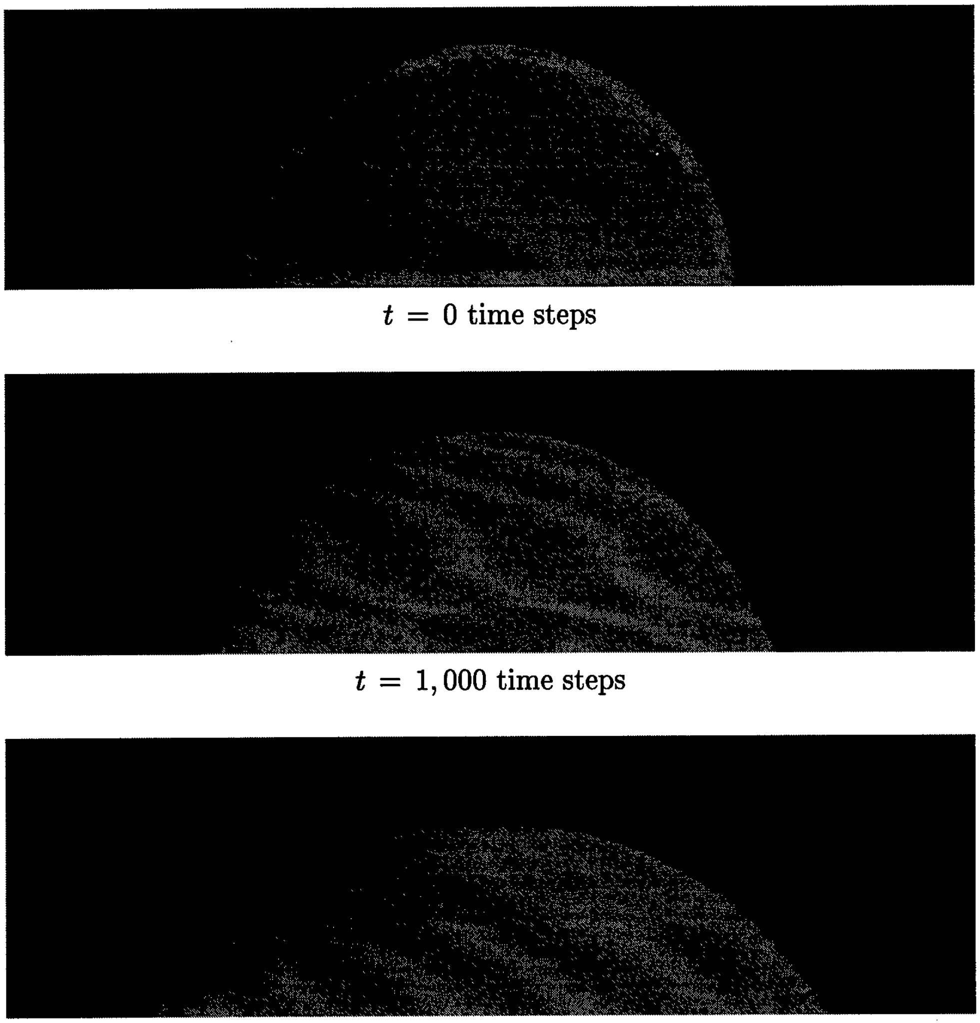
Figure 3.1 shows the time evolution of a sessile drop which wets the bottom wall and is subjected to an acceleration directed downwards ($g = -0.00001$). The drop spreads across the wall surface. Because the lattice was not large enough, the drop ends join laterally because of the periodic boundary conditions we used in the X direction, so that a flat liquid surface is recovered in the final state.

Figure 3.2 shows the equilibrium shape of a nonwetting drop subjected to different values of the gravitational acceleration. One can see that the drop becomes more and more flattened when the value of the gravitational acceleration increases.

3.2 Contact angle

We used a 1024×192 lattice. The necessary CPU time to perform 100,000 time steps was approx. 72 hours for each data set. Several values of the surface tension coefficient κ were considered, while the normal derivative of the chemical potential on the lower wall surface was $\frac{\partial \mu}{\partial x_n} = 0.01$. To account only for the effect of surface tension on the contact angle, no gravitational acceleration was considered ($\mathbf{g} = 0$).

Figures 3.3 - 3.6 show the time evolution of the lattice state for different values of the surface tension coefficient κ. When the value of the surface coefficient is large enough ($\kappa = 0.01$), the drop spreads along the whole lattice domain, so that the contact angle cannot be determined because of the periodic boundary conditions on the left and right margin of the lattice (Figure 3.3). For smaller values of the surface tension, coefficient, when the drop does not spread along the whole bottom wall, the contact angle α may be determined using the following procedure (Figure 3.7), which takes into account the fact that the upper drop border becomes a circular arc when the equilibrium configuration is reached (i.,e., when the number of time steps is large enough). The drop height is denoted h, while its width is denoted w.



$t = 0$ time steps

$t = 1,000$ time steps

$t = 2,000$ time steps

Figure 3.1: Time evolution of the shape of a wetting drop.

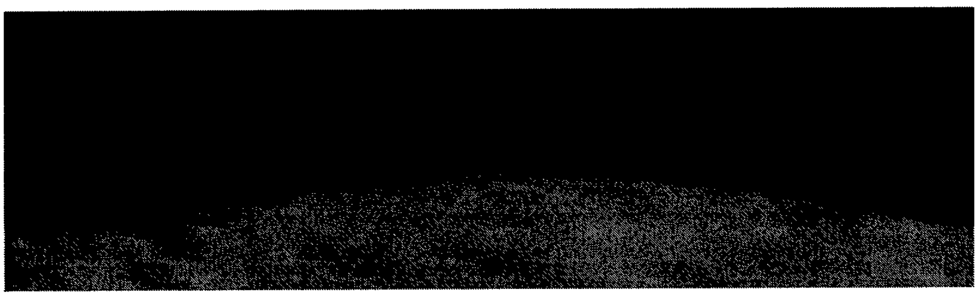
$t = 3,000$ time steps

$t = 4,000$ time steps

$t = 5,000$ time steps

Figure 3.1: *(continued)* Time evolution of the shape of a wetting drop.

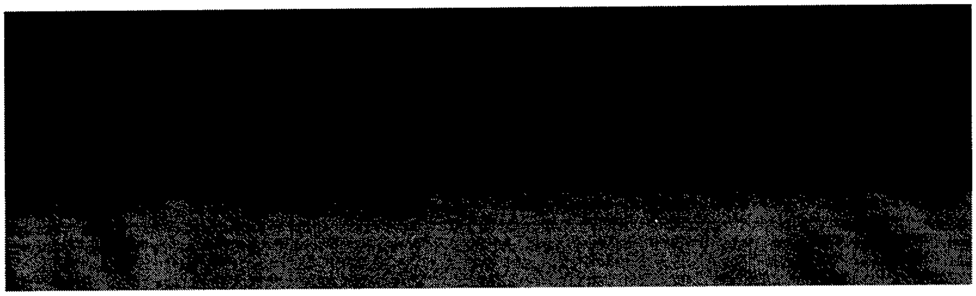
$t = 6,000$ time steps



$t = 7,000$ time steps

$t = 10,000$ time steps

Figure 3.1: *(continued)* Time evolution of the shape of a wetting drop.



$t = 30,000$ time steps

Figure 3.1: *(continued)* Time evolution of the shape of a wetting drop.

If R is the arc radius, we have

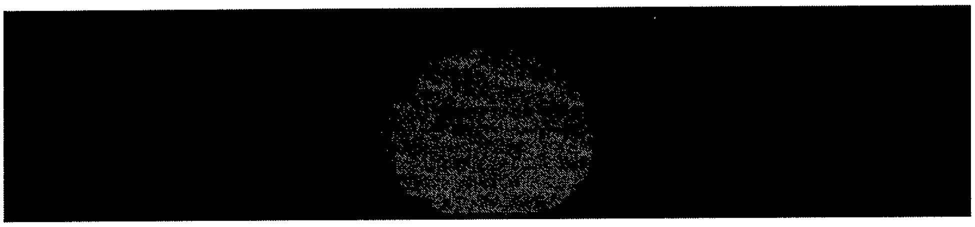
$$h = R(1 - \cos\alpha) \tag{3.3}$$

$$w = 2R\sin\alpha \tag{3.4}$$

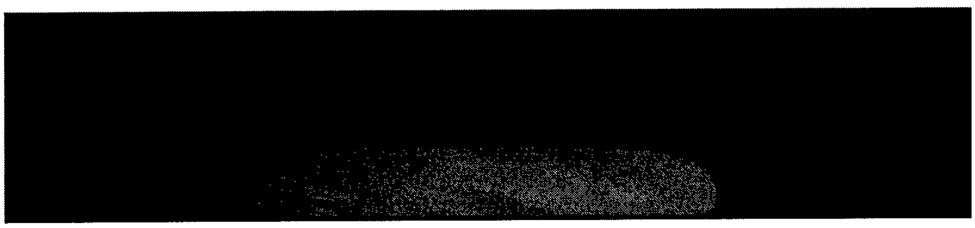
$$\frac{h}{w} = \frac{1 - \cos\alpha}{2\sin\alpha} = \frac{1}{2}\tan\frac{\alpha}{2} \tag{3.5}$$

$$\cos\alpha = 2\arctan\left(\frac{2h}{w}\right) \tag{3.6}$$

Table 3.1 shows the values of the contact angles for the cases shown in figures 3.4 - 3.6.



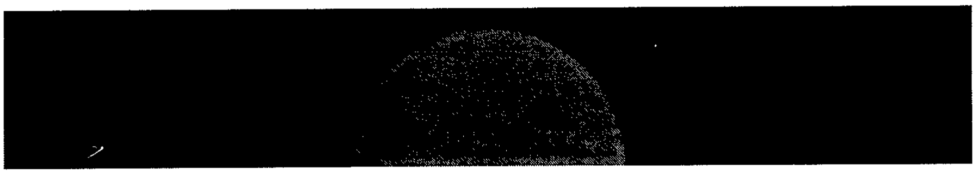
$g = -0.000001$



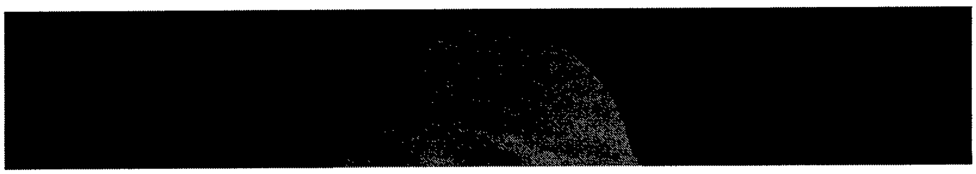
$g = -0.000020$

$g = -0.000050$

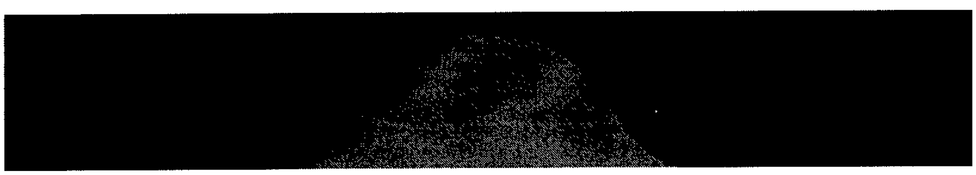
Figure 3.2: Shape of a non - wetting drop subjected to different values of the gravitational field.



initial state ($t = 0$)



$t = 1,000$ time steps

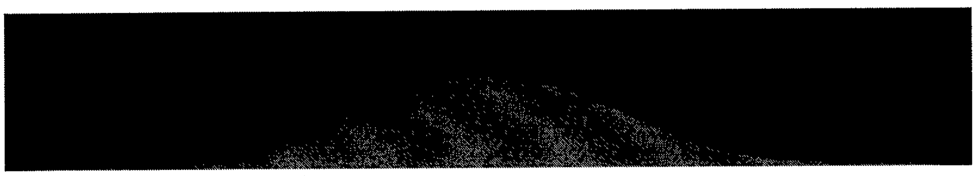


$t = 5,000$ time steps

$t = 10,000$ time steps

Figure 3.3: Drop shape during simulation with $\kappa = 0.0100$.

$t = 20,000$ time steps



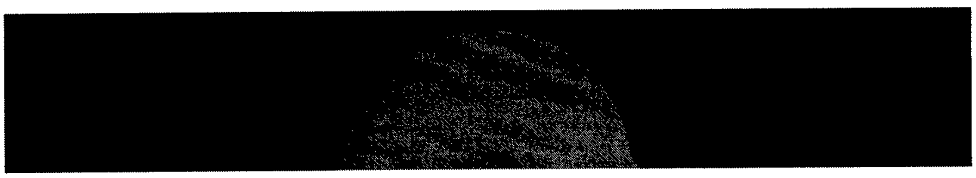
$t = 30,000$ time steps

$t = 32,000$ time steps

$t = 33,000$ time steps

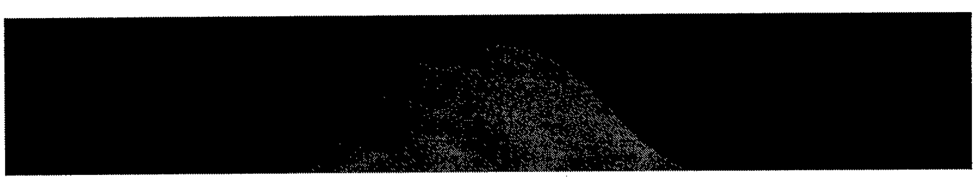
Figure 3.3: *(continued)* Drop shape during simulation with $\kappa = 0.0100$.

initial state ($t = 0$)



$t = 1,000$ time steps

$t = 5,000$ time steps



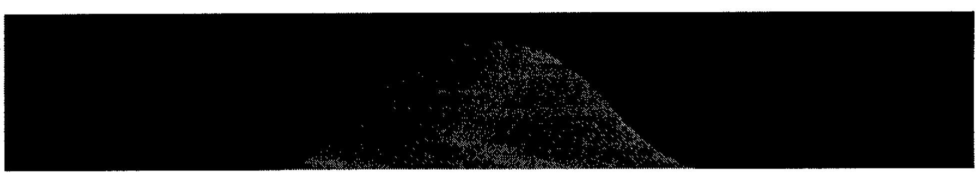
$t = 10,000$ time steps

Figure 3.4: Drop shape during simulation with $\kappa = 0.0094$.

$t = 20,000$ time steps

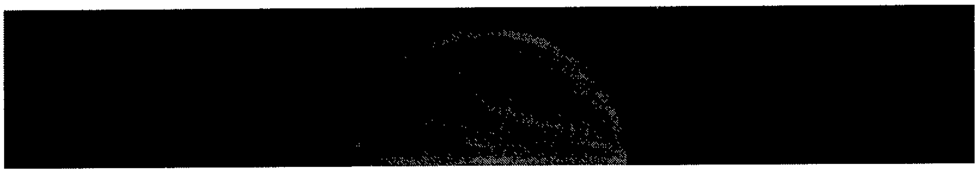
$t = 30,000$ time steps

$t = 50,000$ time steps

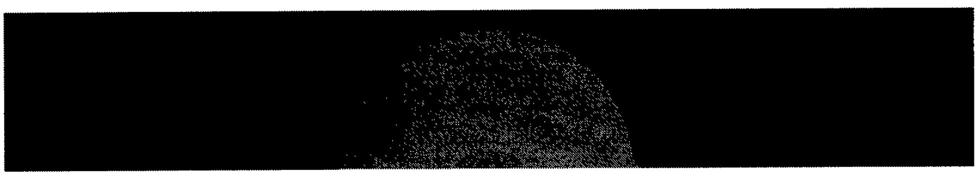


$t = 100,000$ time steps

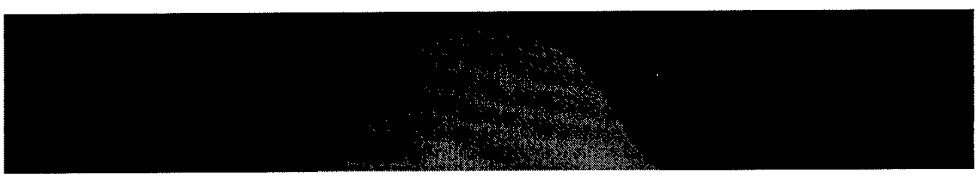
Figure 3.4: *(continued)* Drop shape during simulation with $\kappa = 0.0094$.



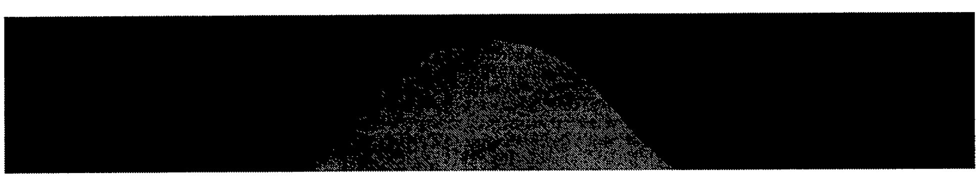
initial state ($t = 0$)



$t = 1,000$ time steps



$t = 5,000$ time steps



$t = 10,000$ time steps

Figure 3.5: Drop shape during simulation with $\kappa = 0.0090$.

t = 20,000 time steps

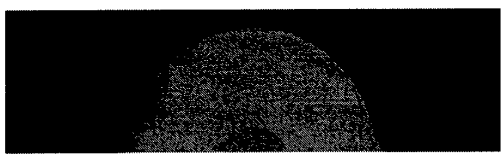
t = 30,000 time steps

t = 50,000 time steps

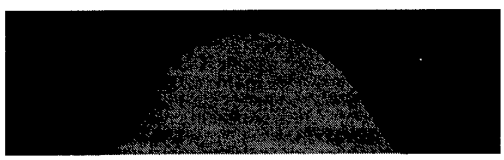
t = 100,000 time steps

Figure 3.5: *(continued)* Drop shape during simulation with $\kappa = 0.0090$.

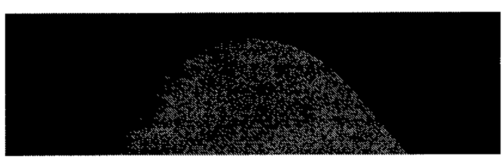
initial state ($t = 0$)



$t = 1,000$ time steps



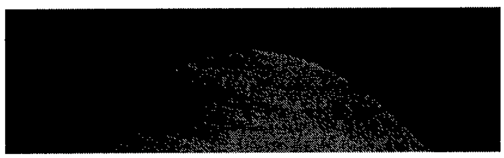
$t = 5,000$ time steps



$t = 10,000$ time steps

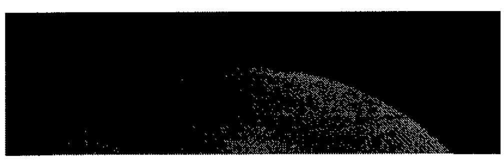
Figure 3.6: Drop shape during simulation with $\kappa = 0.0088$.

$t = 20,000$ time steps



$t = 30,000$ time steps

$t = 50,000$ time steps



$t = 100,000$ time steps

Figure 3.6: *(continued)* Drop shape during simulation with $\kappa = 0.0088$.

Table 3.1: Influence of the surface tension coefficient κ on the contact angle.

κ	$\cos\alpha$
0.0094	0.9097
0.0090	0.7625
0.0088	0.6671

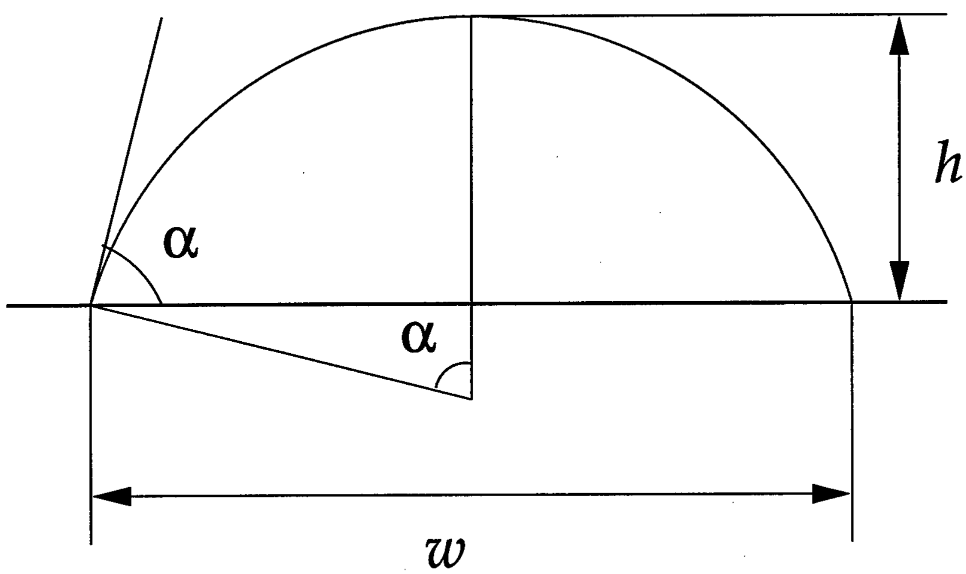


Figure 3.7: Determination of the value of the contact angle α.

References

[1] V. SOFONEA, *Lattice Boltzmann Models for Multicomponent Fluids*, Intermediate Report #002, Contract SPC – 98 – 4061 No. F61775 – 98 – WE101, EOARD (January 1999).

[2] W. H. PRESS, S. A. TEUKOLSKY, W. T. Vetterling, B. P. FLANNERY, *Numerical Recipes in Fortran: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1992.

[3] C. HIRSCH, *Numerical Computation of Internal and External Flows, volume I: Fundamentals of Numerical Discretization*, John Wiley and Sons, Chichester, New York, 1988.

[4] K. W. MORTON, D. F. MAYERS, *Numerical Solutions of Partial Differential Equations*, Cambridge University Press, Cambridge, 1996.

[5] X. HE, L. S. LUO, M. DEMBO, *Some Progress in Lattice Boltzmann Method. Part I. Nonuniform Mesh Grids*, Journal of Computational Physics **129** (1996) 357.

[6] X. HE, G. D. DOOLEN, *Lattice Boltzmann method on a curvilinear coordinate system: Vortex shedding behind a circular cylinder*, Physical Review **E56** (1997) 434.

[7] X. HE, L. S. LUO, M. DEMBO, *Some Progress in Lattice Boltzmann Method: Reynolds number enhancement in simulations*, Physica A **239** (1997) 276.

[8] X. HE, *Error analysis for the interpolation - supplemented lattice Boltzmann equation scheme*, International Journal of Modern Physics **C8** (1997) 737.

[9] K. H. HUEBNER, *The Finite Element Method for Engineers*, John Wiley and Sons, New York, 1975.

[10] X. Y. HE, L. S. LUO, *Lattice Boltzmann model for the incompressible Navier - Stokes Equation*, Journal of Statistical Physics **88** (1997) 927.

[11] D. H. ROTHMAN, S. ZALESKI, *Lattice - Gas Cellular Automata, Simple Models of Complex Hydrodynamics*, Cambridge University Press, Cambridge, 1997.

[12] P. S. PERERA, R. F. SEKERKA, *Nonsolenoidal flow in a liquid diffusion couple*, Physica of FLuids bf9 (1997) 376.

[13] X. SHAN, D. DOOLEN, *Diffusion in a multicomponent lattice Boltzmann equation model*, Physical Review **E54** (1996) 3614.

[14] X. SHAN, H. CHEN, *Lattice Boltzmann Model for Simulating Flows with Multiple Phases and Components*, Physical Review **E47** (1993) 1815.

[15] V. SOFONEA, *Lattice Boltzmann Approach to Collective Particle Interactions in Magnetic Fluids*, Europhysics Letters **25** (1994) 385.

[16] V. SOFONEA, *Lattice Boltzmann Model for Magnetic Fluids*, Romanian Reports in Physics **47** (1995) 307.

[17] X. SHAN, G. DOOLEN, *Multicomponent Lattice - Boltzmann model with interparticle interaction*, Journal of Statistical Physics **81** (1995) 379.

[18] B. CHOPARD, M. DROZ, *Cellular Automata Modeling of Physical Systems*, Cambridge University Press, Cambridge, 1998.

[19] R. F. SEKERKA *(Principal Investigator)*: Grant NAG3 – 1875, *Fluid Physics in a Stochastic Acceleration Environment: Diffusion and Non – Solenoidal Flows*, Carnegie Mellon University, Pittsburgh, Pennsylvania; *Fluid Physics in a Stochastic Acceleration Environment: A Lattice Boltzmann Approach* (Progress Report, Carnegie Mellon University, February 1999).

[20] X. HE, L. S. LUO, *Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation*, Physical Review **E56** (1997) 6811.

[21] X. SHAN, X. HE, *Discretization of the Velocity Space in the Solution of the Boltzmann Equation*, Physical Review Letters **80** (1998) 65.

[22] X. HE, X. SHAN, G. D. DOOLEN, *Discrete Boltzmann equation model for nonideal gases*, Physical Review **E57** (1998) R13.

[23] D. ROTHMAN, S. ZALESKI, *Lattice - gas models of phase separation: interfaces, phase transitions and multiphase flow*, Reviews of Modern Physics **66** (1994) 1417.

[24] M. R. SWIFT, W. R. OSBORN, J. M. YEOMANS, *Lattice Boltzmann Simulation of Non-Ideal Fluids*, Physical Review Letters **75** (1995) 830.

[25] W. R. OSBORN, E. ORLANDINI, M. R. SWIFT, J. M. YEOMANS, J. R. BANAVAR, *Lattice Boltzmann Study of Hydrodynamic Spinodal Decomposition*, Physical Review Letters **75** (1995) 4031.

[26] E. ORLANDINI, M. R. SWIFT, J. M. YEOMANS, *Lattice Boltzmann Model of Binary Fluid Mixtures*, Europhysics Letters **32** (1995) 463.

[27] M. SWIFT, E. ORLANDINI, W. R. OSBORN, J. YEOMANS, *Lattice Boltzmann simulations of liquid - gas and binary fluid systems*, Physical Review **E54** (1996) 5041.

[28] G. GONNELLA, E. ORLANDINI, J. M. YEOMANS, *Lattice Boltzmann simulations of lamellar and droplet phases*, Physical Review **E58** (1998) 480.

[29] K. R. MECKE, V. SOFONEA, *Morphology of spinodal decomposition*, Physical Review **E56** (1997) R3761 – 3764.

[30] V. SOFONEA, K. R. MECKE, *Morphological Characterization of Spinodal Decomposition Kinetics*, European Physical Journal **B8** (1999) 99 – 112.

[31] V. SOFONEA, *Lattice Boltzmann Models for Multicomponent Fluids*, Preliminary Report #001, Contract SPC – 98 – 4061 No. F61775 – 98 – WE101, EOARD (August 1998).

Appendix A

wet9 code

```c
/*******************************************************************\
*                                                                   *
*  wet9head.h  -- definition of global variables                    *
*                                                                   *
\*******************************************************************/

#ifdef MAIN_HEADER

  const double kboltz=1.381e-23, amu=1.661e-27, temp=300.00;

/*  kboltz = J/(molecules*K), amu = (kg), temp = (K);  */

  const double w0 = ((double) 4) / ((double) 9),
               w1 = ((double) 1) / ((double) 9),
               w2 = ((double) 1) / ((double) 36);

  const double three = ((double) 3),
               three_over_two = ((double) 3) / ((double) 2),
               nine_over_two = ((double) 9) / ((double) 2);

  double uxwall_top, uywall_top, uxwall_bot, uywall_bot;

  double uxwall_left, uywall_left, uxwall_right, uywall_right;

  double *a_uxwall_top, *a_uywall_top, *a_uxwall_bot, *a_uywall_bot;

  double *a_uxwall_left, *a_uywall_left, *a_uxwall_right, *a_uywall_right;

  int nnodes_x, nnodes_y, nnodes_all, lambda;

  int *a_nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda;

  int key_init, key_boundary, key_point, key_scheme, key_interface;

  int *a_key_init, *a_key_boundary, *a_key_scheme, *a_key_interface;

  int nsim, ncycles, niter_cycle, niter_init, iter, niter;

  int *a_ncycles, *a_niter_cycle, *a_niter_init;

  double length_x, length_y, delta_x, delta_y, delta_t, ctau1, ctau2,
         cspeed1, cspeed2, cspeed12, cspeed22, cfl1, cfl2,
         force_x, force_y, csforce_x, csforce_y;

  double gforce, kforce, kbound1, kbound2;

  double oneminus1, oneminus2, oneplus1, oneplus2;

  double t1minus, t1center, t1plus;
  double t2minus, t2center, t2plus;
  double t1fminus1, t1fplus1, t1fminus2, t1fplus2;

  double in1, in2, in1plus1, in2plus1, in1plus2, in2plus2;
  double up1, up2, up1minus1, up2minus1, up1minus2, up2minus2;
  double tin1, tin1plus1, tin1plus2, tout1, tout1minus1, tout1minus2;
  double tin2, tin2plus1, tin2plus2, tout2, tout2minus1, tout2minus2;
  double isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2;

  double *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_t,
         *a_force_x, *a_force_y, *a_gforce;

  double mass1, mass2, tau1, tau2;

  double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2;

  double nzero1, nzero2, nzero1left, nzero1right, nzero2left, nzero2right;

  double *a_nzero1, *a_nzero2, *a_nzero1left, *a_nzero2left,
         *a_nzero1right, *a_nzero2right;
```



```c
  char input_name[] = "wet9.input", output_name[] = "wet9.output";

  char id_name[128], rez_name[128], xv_name[128];

  double ww[9], ecx[9], ecy[9], ecx1[9], ecy1[9], ecx2[9], ecy2[9];

  double nc10[9], nc11[9], nc12[9], nc13[9], nc14[9], nc15[9],
         nc16[9], nc17[9], nc18[9];

  double nc20[9], nc21[9], nc22[9], nc23[9], nc24[9], nc25[9],
         nc26[9], nc27[9], nc28[9];

  double ncbot10[9], ncbot11[9], ncbot12[9], ncbot13[9], ncbot14[9], ncbot15[9],
         ncbot16[9], ncbot17[9], ncbot18[9];

  double ncbot20[9], ncbot21[9], ncbot22[9], ncbot23[9], ncbot24[9], ncbot25[9],
         ncbot26[9], ncbot27[9], ncbot28[9];

  double nctop10[9], nctop11[9], nctop12[9], nctop13[9], nctop14[9], nctop15[9],
         nctop16[9], nctop17[9], nctop18[9];

  double nctop20[9], nctop21[9], nctop22[9], nctop23[9], nctop24[9], nctop25[9],
         nctop26[9], nctop27[9], nctop28[9];

  double ncleft10[9], ncleft11[9], ncleft12[9], ncleft13[9], ncleft14[9],
         ncleft15[9], ncleft16[9], ncleft17[9], ncleft18[9];

  double ncleft20[9], ncleft21[9], ncleft22[9], ncleft23[9], ncleft24[9],
         ncleft25[9], ncleft26[9], ncleft27[9], ncleft28[9];

  double ncright10[9], ncright11[9], ncright12[9], ncright13[9], ncright14[9],
         ncright15[9], ncright16[9], ncright17[9], ncright18[9];

  double ncright20[9], ncright21[9], ncright22[9], ncright23[9], ncright24[9],
         ncright25[9], ncright26[9], ncright27[9], ncright28[9];

  double nu10[9], nu11[9], nu12[9], nu13[9], nu14[9], nu15[9],
         nu16[9], nu17[9], nu18[9];

  double nu20[9], nu21[9], nu22[9], nu23[9], nu24[9], nu25[9],
         nu26[9], nu27[9], nu28[9];

  double ns1tl[9], ns1tr[9], ns1bl[9], ns1br[9],
         ns2tl[9], ns2tr[9], ns2bl[9], ns2br[9];

  double nsb1tl[9], nsb1tr[9], nsb1bl[9], nsb1br[9],
         nsb2tl[9], nsb2tr[9], nsb2bl[9], nsb2br[9];

  double nst1tl[9], nst1tr[9], nst1bl[9], nst1br[9],
         nst2tl[9], nst2tr[9], nst2bl[9], nst2br[9];

  double nsl1tl[9], nsl1tr[9], nsl1bl[9], nsl1br[9],
         nsl2tl[9], nsl2tr[9], nsl2bl[9], nsl2br[9];

  double nsr1tl[9], nsr1tr[9], nsr1bl[9], nsr1br[9],
         nsr2tl[9], nsr2tr[9], nsr2bl[9], nsr2br[9];

  double nl1tl[9], nl1tr[9], nl1bl[9], nl1br[9],
         nl2tl[9], nl2tr[9], nl2bl[9], nl2br[9];

  double nlb1tl[9], nlb1tr[9], nlb1bl[9], nlb1br[9],
         nlb2tl[9], nlb2tr[9], nlb2bl[9], nlb2br[9];

  double nlt1tl[9], nlt1tr[9], nlt1bl[9], nlt1br[9],
         nlt2tl[9], nlt2tr[9], nlt2bl[9], nlt2br[9];

  double nll1tl[9], nll1tr[9], nll1bl[9], nll1br[9],
         nll2tl[9], nll2tr[9], nll2bl[9], nll2br[9];
```

```c
  double nlr1tl[9], nlr1tr[9], nlr1bl[9], nlr1br[9],
         nlr2tl[9], nlr2tr[9], nlr2bl[9], nlr2br[9];

  double ecxforce1[9], ecyforce1[9], ecxforce2[9], ecyforce2[9],
         ecprod1[9], ecprod2[9];

  int *boundary_mode;

  double *f10, *f11, *f12, *f13, *f14, *f15, *f16, *f17, *f18;

  double *f20, *f21, *f22, *f23, *f24, *f25, *f26, *f27, *f28;

  double *ff10, *ff11, *ff12, *ff13, *ff14, *ff15, *ff16, *ff17, *ff18;

  double *ff20, *ff21, *ff22, *ff23, *ff24, *ff25, *ff26, *ff27, *ff28;

  double *neq10, *neq11, *neq12, *neq13, *neq14, *neq15, *neq16,
         *neq17, *neq18;

  double *neq20, *neq21, *neq22, *neq23, *neq24, *neq25, *neq26,
         *neq27, *neq28;

  double *source10, *source11, *source12, *source13, *source14, *source15,
         *source16, *source17, *source18;

  double *source20, *source21, *source22, *source23, *source24, *source25,
         *source26, *source27, *source28;

  double *sf10, *sf11, *sf12, *sf13, *sf14, *sf15, *sf16, *sf17, *sf18;

  double *sf20, *sf21, *sf22, *sf23, *sf24, *sf25, *sf26, *sf27, *sf28;

  double *uxloc, *uyloc, *uxloc1, *uyloc1, *uxloc2, *uyloc2;

  double *nloc1, *nloc2, *colorfield, *gradcolor_x, *gradcolor_y;

  double *gradn1x, *gradn2x, *gradn1y, *gradn2y;

  int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;

#else

  extern const double kboltz, amu, temp;

  extern const double w0, w1, w2;

  extern const double three, three_over_two, nine_over_two;

  extern double uxwall_top, uywall_top, uxwall_bot, uywall_bot;

  extern double uxwall_left, uywall_left, uxwall_right, uywall_right;

  extern double *a_uxwall_top, *a_uywall_top, *a_uxwall_bot, *a_uywall_bot;

  extern double *a_uxwall_left, *a_uywall_left,
                *a_uxwall_right, *a_uywall_right;

  extern int nnodes_x, nnodes_y, nnodes_all, lambda;

  extern int *a_nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda;

  extern int key_init, key_boundary, key_point, key_scheme, key_interface;

  extern int *a_key_init, *a_key_boundary, *a_key_scheme, *a_key_interface;

  extern int nsim, ncycles, niter_cycle, niter_init, iter, niter;

  extern int *a_ncycles, *a_niter_cycle, *a_niter_init;
```



```c
  extern double length_x, length_y, delta_x, delta_y, delta_t, ctau1, ctau2,
                cspeed1, cspeed2, cspeed12, cspeed22, cfl1, cfl2,
                force_x, force_y, csforce_x, csforce_y;

  extern double gforce, kforce, kbound1, kbound2;

  extern double oneminus1, oneminus2, oneplus1, oneplus2;

  extern double t1minus, t1center, t1plus;
  extern double t2minus, t2center, t2plus;
  extern double t1fminus1, t1fplus1, t1fminus2, t1fplus2;

  extern double in1, in2, in1plus1, in2plus1, in1plus2, in2plus2;
  extern double up1, up2, up1minus1, up2minus1, up1minus2, up2minus2;
  extern double tin1, tin1plus1, tin1plus2, tout1, tout1minus1, tout1minus2;
  extern double tin2, tin2plus1, tin2plus2, tout2, tout2minus1, tout2minus2;
  extern double isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2;

  extern double *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_t,
                *a_force_x, *a_force_y, *a_gforce;

  extern double mass1, mass2, tau1, tau2;

  extern double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2;

  extern double nzero1, nzero2, nzero1left, nzero2left,
    nzero1right, nzero2right;

  extern double *a_nzero1, *a_nzero2, *a_nzero1left, *a_nzero2left,
    *a_nzero1right, *a_nzero2right;

  extern char input_name[], output_name[];

  extern char id_name[], rez_name[], xv_name[];

  extern double ww[], ecx[], ecy[], ecx1[], ecy1[], ecx2[], ecy2[];

  extern double nc10[], nc11[], nc12[], nc13[], nc14[], nc15[],
                nc16[], nc17[], nc18[];

  extern double nc20[], nc21[], nc22[], nc23[], nc24[], nc25[],
                nc26[], nc27[], nc28[];

  extern double ncbot10[9], ncbot11[9], ncbot12[9], ncbot13[9], ncbot14[9],
                ncbot15[9], ncbot16[9], ncbot17[9], ncbot18[9];

  extern double ncbot20[9], ncbot21[9], ncbot22[9], ncbot23[9], ncbot24[9],
                ncbot25[9], ncbot26[9], ncbot27[9], ncbot28[9];

  extern double nctop10[9], nctop11[9], nctop12[9], nctop13[9], nctop14[9],
                nctop15[9], nctop16[9], nctop17[9], nctop18[9];

  extern double nctop20[9], nctop21[9], nctop22[9], nctop23[9], nctop24[9],
                nctop25[9], nctop26[9], nctop27[9], nctop28[9];

  extern double ncleft10[9], ncleft11[9], ncleft12[9], ncleft13[9], ncleft14[9],
                ncleft15[9], ncleft16[9], ncleft17[9], ncleft18[9];

  extern double ncleft20[9], ncleft21[9], ncleft22[9], ncleft23[9], ncleft24[9],
                ncleft25[9], ncleft26[9], ncleft27[9], ncleft28[9];

  extern double ncright10[9], ncright11[9], ncright12[9], ncright13[9],
                ncright14[9], ncright15[9], ncright16[9], ncright17[9],
                ncright18[9];

  extern double ncright20[9], ncright21[9], ncright22[9], ncright23[9],
                ncright24[9], ncright25[9], ncright26[9], ncright27[9],
                ncright28[9];
```

```
extern double nu10[], nu11[], nu12[], nu13[], nu14[], nu15[],
              nu16[], nu17[], nu18[];

extern double nu20[], nu21[], nu22[], nu23[], nu24[], nu25[],
              nu26[], nu27[], nu28[];

extern double ns1tl[], ns1tr[], ns1bl[], ns1br[],
              ns2tl[], ns2tr[], ns2bl[], ns2br[];

extern double nsb1tl[], nsb1tr[], nsb1bl[], nsb1br[],
              nsb2tl[], nsb2tr[], nsb2bl[], nsb2br[];

extern double nst1tl[], nst1tr[], nst1bl[], nst1br[],
              nst2tl[], nst2tr[], nst2bl[], nst2br[];

extern double nsl1tl[], nsl1tr[], nsl1bl[], nsl1br[],
              nsl2tl[], nsl2tr[], nsl2bl[], nsl2br[];

extern double nsr1tl[], nsr1tr[], nsr1bl[], nsr1br[],
              nsr2tl[], nsr2tr[], nsr2bl[], nsr2br[];

extern double nl1tl[], nl1tr[], nl1bl[], nl1br[],
              nl2tl[], nl2tr[], nl2bl[], nl2br[];

extern double nlb1tl[], nlb1tr[], nlb1bl[], nlb1br[],
              nlb2tl[], nlb2tr[], nlb2bl[], nlb2br[];

extern double nlt1tl[], nlt1tr[], nlt1bl[], nlt1br[],
              nlt2tl[], nlt2tr[], nlt2bl[], nlt2br[];

extern double nll1tl[], nll1tr[], nll1bl[], nll1br[],
              nll2tl[], nll2tr[], nll2bl[], nll2br[];

extern double nlr1tl[], nlr1tr[], nlr1bl[], nlr1br[],
              nlr2tl[], nlr2tr[], nlr2bl[], nlr2br[];

extern double ecxforce1[], ecyforce1[], ecxforce2[], ecyforce2[],
              ecprod1[], ecprod2[];

extern int *boundary_mode;

extern double *f10, *f11, *f12, *f13, *f14, *f15, *f16, *f17, *f18;

extern double *f20, *f21, *f22, *f23, *f24, *f25, *f26, *f27, *f28;

extern double *ff10, *ff11, *ff12, *ff13, *ff14, *ff15, *ff16,
              *ff17, *ff18;

extern double *ff20, *ff21, *ff22, *ff23, *ff24, *ff25, *ff26,
              *ff27, *ff28;

extern double *neq10, *neq11, *neq12, *neq13, *neq14, *neq15, *neq16,
              *neq17, *neq18;

extern double *neq20, *neq21, *neq22, *neq23, *neq24, *neq25, *neq26,
              *neq27, *neq28;

extern double *source10, *source11, *source12, *source13, *source14,
              *source15, *source16, *source17, *source18;

extern double *source20, *source21, *source22, *source23, *source24,
              *source25, *source26, *source27, *source28;

extern double *sf10, *sf11, *sf12, *sf13, *sf14, *sf15, *sf16, *sf17, *sf18;

extern double *sf20, *sf21, *sf22, *sf23, *sf24, *sf25, *sf26, *sf27, *sf28;

extern double *uxloc, *uyloc, *uxloc1, *uyloc1, *uxloc2, *uyloc2; ;
```



```
extern double *nloc1, *nloc2, *colorfield, *gradcolor_x, *gradcolor_y;

extern double *gradn1x, *gradn2x, *gradn1y, *gradn2y;

extern int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;

#endif
```

```c
/**************************************************************************\
*                                                                          *
*  wet9inout.c                                                             *
*                                                                          *
\**************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "wet9head.h"

void allocate_nsim(void)
{
  a_nnodes_x = (int*) malloc(nsim*sizeof(int));
  a_nnodes_y = (int*) malloc(nsim*sizeof(int));
  a_nnodes_all = (int*) malloc(nsim*sizeof(int));
  a_lambda = (int*) malloc(nsim*sizeof(int));
  a_key_init = (int*) malloc(nsim*sizeof(int));
  a_key_boundary = (int*) malloc(nsim*sizeof(int));
  a_key_scheme = (int*) malloc(nsim*sizeof(int));
  a_key_interface = (int*) malloc(nsim*sizeof(int));
  a_ncycles = (int*) malloc(nsim*sizeof(int));
  a_niter_cycle = (int*) malloc(nsim*sizeof(int));
  a_niter_init = (int*) malloc(nsim*sizeof(int));
  a_length_x = (double*) malloc(nsim*sizeof(double));
  a_length_y = (double*) malloc(nsim*sizeof(double));
  a_delta_x = (double*) malloc(nsim*sizeof(double));
  a_delta_y = (double*) malloc(nsim*sizeof(double));
  a_delta_t = (double*) malloc(nsim*sizeof(double));
  a_force_x = (double*) malloc(nsim*sizeof(double));
  a_force_y = (double*) malloc(nsim*sizeof(double));
  a_gforce = (double*) malloc(nsim*sizeof(double));
  a_mass1 = (double*) malloc(nsim*sizeof(double));
  a_mass2 = (double*) malloc(nsim*sizeof(double));
  a_cspeed1 = (double*) malloc(nsim*sizeof(double));
  a_cspeed2 = (double*) malloc(nsim*sizeof(double));
  a_tau1 =  (double*) malloc(nsim*sizeof(double));
  a_tau2 =  (double*) malloc(nsim*sizeof(double));
  a_nzero1 =  (double*) malloc(nsim*sizeof(double));
  a_nzero2 =  (double*) malloc(nsim*sizeof(double));
  a_nzero1left =  (double*) malloc(nsim*sizeof(double));
  a_nzero2left =  (double*) malloc(nsim*sizeof(double));
  a_nzero1right =  (double*) malloc(nsim*sizeof(double));
  a_nzero2right =  (double*) malloc(nsim*sizeof(double));
  a_uxwall_top =  (double*) malloc(nsim*sizeof(double));
  a_uywall_top =  (double*) malloc(nsim*sizeof(double));
  a_uxwall_bot =  (double*) malloc(nsim*sizeof(double));
  a_uywall_bot =  (double*) malloc(nsim*sizeof(double));
  a_uxwall_left =  (double*) malloc(nsim*sizeof(double));
  a_uywall_left =  (double*) malloc(nsim*sizeof(double));
  a_uxwall_right =  (double*) malloc(nsim*sizeof(double));
  a_uywall_right =  (double*) malloc(nsim*sizeof(double));
}

void free_nsim(void)
{
  free(a_nnodes_x);
  free(a_nnodes_y);
  free(a_nnodes_all);
  free(a_lambda);
  free(a_key_init);
  free(a_key_boundary);
  free(a_key_scheme);
  free(a_key_interface);
  free(a_ncycles);
  free(a_niter_cycle);
  free(a_niter_init);
```



```c
  free(a_length_x);
  free(a_length_y);
  free(a_delta_x);
  free(a_delta_y);
  free(a_delta_t);
  free(a_force_x);
  free(a_force_y);
  free(a_gforce);
  free(a_mass1);
  free(a_mass2);
  free(a_cspeed1);
  free(a_cspeed2);
  free(a_tau1);
  free(a_tau2);
  free(a_nzero1);
  free(a_nzero2);
  free(a_nzero1left);
  free(a_nzero2left);
  free(a_nzero1right);
  free(a_nzero2right);
  free(a_uxwall_top);
  free(a_uywall_top);
  free(a_uxwall_bot);
  free(a_uywall_bot);
  free(a_uxwall_left);
  free(a_uywall_left);
  free(a_uxwall_right);
  free(a_uywall_right);
}

void wet9_input(void)
{
  int i, j;
  char dummy[128];
  /*
    double kboltz=1.381e-23, amu=1.661e-27, temp=300.00;
    kboltz = J/(molecules*K), amu = (kg), temp - (K);
  */
  FILE *fin, *frez;

  frez=fopen(output_name,"w");
  if((fin = fopen(input_name,"r")) == NULL)
  {
    fprintf(frez,"program <wet9> stopped - input file <%s> does not exist !\n",
            input_name);
    fclose(frez);
    exit(1);
  }
  fscanf(fin,"%s %d\n",dummy,&nsim);
  if(nsim < 1)
    exit(1);

  allocate_nsim();

  for(i=0; i<nsim; i++)
    {
      fscanf(fin,"%s %d %s %d\n",dummy,&nnodes_x,dummy,&nnodes_y);
      fprintf(frez,"nnodes_x= %d  nnodes_y= %d\n",nnodes_x,nnodes_y);
      fscanf(fin,"%s %d\n",dummy,&lambda);
      fprintf(frez,"lambda= %d\n",lambda);
      fscanf(fin,"%s %lf %s %lf\n",dummy,&delta_x,dummy,&delta_y);
      if(delta_y != delta_x)
        delta_y = delta_x;
      fprintf(frez,"delta_x= %lf  delta_y= %lf\n",delta_x,delta_y);
      fscanf(fin,"%s %lf\n",dummy,&delta_t);
      fprintf(frez,"delta_t= %20.15lf\n",delta_t);
      fscanf(fin,"%s %d %s %d\n",dummy,&key_init,dummy,&key_boundary);
      fprintf(frez,"key_init= %d  key_boundary= %d\n",key_init,key_boundary);
      fscanf(fin,"%s %d %s %d\n",dummy,&key_scheme,dummy,&key_interface);
```

```
printf("key_interface=%d\n",key_interface);
fprintf(frez,"key_scheme=%d\n",key_scheme);
fscanf(fin,"%s %d %s %d %s %d\n",
      dummy,&ncycles,dummy,&niter_cycle,dummy,&niter_init);
fprintf(frez,"ncycles= %d  niter_cycle= %d  niter_init= %d\n",
      ncycles,niter_cycle,niter_init);
fscanf(fin,"%s %lf %s %lf\n",dummy,&mass1,dummy,&tau1);
fscanf(fin,"%s %lf %s %lf\n",dummy,&mass2,dummy,&tau2);
fscanf(fin,"%s %lf %s %lf\n",dummy,&nzero1left,dummy,&nzero2left);
fscanf(fin,"%s %lf %s %lf\n",dummy,&nzero1right,dummy,&nzero2right);
nzero1 = nzero1left;
nzero2 = nzero2left;
fprintf(frez,"mass1=%lf  tau1=%e\n",mass1,tau1);
fprintf(frez,"mass2=%lf  tau2=%e\n",mass2,tau2);
fprintf(frez,"nzero1=%lf  nzero2=%lf\n",nzero1,nzero2);
fprintf(frez,"nzero1left=%lf  nzero2right=%lf\n",
        nzero1left,nzero2left);
fprintf(frez,"nzero1right=%lf  nzero2right=%lf\n",
        nzero1right,nzero2right);
fscanf(fin,"%s %lf %s %lf\n",dummy,&uxwall_bot,dummy,&uywall_bot);
fprintf(frez,"uxwall_bot=%lf  uywall_bot=%lf\n",uxwall_bot,uywall_bot);
fscanf(fin,"%s %lf %s %lf\n",dummy,&uxwall_top,dummy,&uywall_top);
fprintf(frez,"uxwall_top=%lf  uywall_top=%lf\n",uxwall_top,uywall_top);
fscanf(fin,"%s %lf %s %lf\n",dummy,&uxwall_left,dummy,&uywall_left);
fprintf(frez,"uxwall_left=%lf  uywall_left=%lf\n",
        uxwall_left,uywall_left);
fscanf(fin,"%s %lf %s %lf\n",dummy,&uxwall_right,dummy,&uywall_right);
fprintf(frez,"uxwall_right=%lf  uywall_right=%lf\n",
        uxwall_right,uywall_right);
fscanf(fin,"%s %lf %s %lf\n",dummy,&force_x,dummy,&force_y,
      dummy,&gforce);
force_x *= 1.0e-20;
force_y *= 1.0e-20;
gforce *= 1.0e-20;
fprintf(frez,"force_x=%e  force_y=%e  gforce=%e\n",
        force_x,force_y,gforce);

if(mass1)
  cspeed1 = (3.00 * kboltz * temp) / (mass1 * amu);
else
  cspeed1 = 1.0000;
if(mass2)
  cspeed2 = (3.00 * kboltz * temp) / (mass2 * amu);
else
  cspeed2 = 1.0000;
cspeed1 = sqrt(cspeed1);
cspeed2 = sqrt(cspeed2);
fprintf(frez,"cspeed1=%25.20e  cspeed2=%25.20e\n",cspeed1,cspeed2);

/*
if(key_scheme == 6)
  {
  delta_x = delta_y = cspeed1 * tau1;
  if(cspeed2*tau2 > delta_x)
    {
    delta_x =delta_y = cspeed2 * tau2;
    }
  fprintf(frez,"key_scheme=%d  delta_x= %lf  delta_y= %lf\n",
          key_scheme,delta_x,delta_y);
  }
*/
/*
if(( (delta_t*cspeed1 < delta_x) ||
    (delta_t*cspeed1 < delta_y) ||
    (delta_t*cspeed2 < delta_x) ||
    (delta_t*cspeed2 < delta_y)) && (key_scheme < 6) )
  {
    printf("<wet9> stopped - large delta_x or delta_y !\n");
    fprintf(frez,"<wet9> stopped - large delta_x or delta_y !\n");
```



```
      fclose(frez);
      exit(1);
    }
  */
  /*
  if( ((delta_t > tau1) || (delta_t > tau2) && (key_scheme < 6)) )
    {
      printf("<wet9> stopped - delta_t > tau 1 or tau2 !\n");
      fprintf(frez,"<wet9> stopped - delta_t > tau 1 or tau2 !\n");
      fclose(frez);
      exit(1);
    }
  */
  /*
  if(cspeed1 > cspeed2)
    delta_t = delta_x / cspeed1;
  else
    delta_t = delta_x / cspeed2;
  */
  nnodes_all = nnodes_x * nnodes_y;
  length_x = delta_x * ((double) (nnodes_x-1));
  length_y = delta_y * ((double) (nnodes_y-1));

  fprintf(frez,"%d  %d  %d  %lf  %lf  %e\n",nnodes_x,nnodes_y,
          nnodes_all,length_x,length_y,delta_t);
  fprintf(frez,"kboltz= %e  %lf %lf %lf %e\n",kboltz,temp,mass1,mass2,amu);

  a_nnodes_x[i] = nnodes_x;
  a_nnodes_y[i] = nnodes_y;
  a_nnodes_all[i] = nnodes_all;
  a_lambda[i] = lambda;
  a_key_init[i] = key_init;
  a_key_boundary[i] = key_boundary;
  a_key_scheme[i] = key_scheme;
  a_key_interface[i] = key_interface;
  a_ncycles[i] = ncycles;
  a_niter_cycle[i] = niter_cycle;
  a_niter_init[i] = niter_init;
  a_length_x[i] = length_x;
  a_length_y[i] = length_y;
  a_delta_x[i] = delta_x;
  a_delta_y[i] = delta_y;
  a_delta_t[i] = delta_t;
  a_force_x[i] = force_x;
  a_force_y[i] = force_y;
  a_gforce[i] = gforce;
  a_mass1[i] = mass1;
  a_mass2[i] = mass2;
  a_cspeed1[i] = cspeed1;
  a_cspeed2[i] = cspeed2;
  a_tau1[i] = tau1;
  a_tau2[i] = tau2;
  a_nzero1[i] = nzero1;
  a_nzero2[i] = nzero2;
  a_nzero1left[i] = nzero1left;
  a_nzero2left[i] = nzero2left;
  a_nzero1right[i] = nzero1right;
  a_nzero2right[i] = nzero2right;
  a_uxwall_bot[i] = uxwall_bot;
  a_uywall_bot[i] = uywall_bot;
  a_uxwall_top[i] = uxwall_top;
  a_uywall_top[i] = uywall_top;
  a_uxwall_left[i] = uxwall_left;
  a_uywall_left[i] = uywall_left;
  a_uxwall_right[i] = uxwall_right;
  a_uywall_right[i] = uywall_right;
  }
fclose(fin);
fclose(frez);
```

```
}

void build_names()
{
  int imass1,imass2,itau1,itau2,icspeed1,icspeed2,
    idelta_x,idelta_y,idelta_t,iforce_x,iforce_y,inzero1,inzero2;
  imass1 = (int) (mass1 * 100);
  imass2 = (int) (mass2 * 100);
  itau1 = (int) (tau1 * 1.00e+12);
  itau2 = (int) (tau2 * 1.00e+12);
  icspeed1 = (int) cspeed1;
  icspeed2 = (int) cspeed2;
  idelta_x = (int) (delta_x * 1.00e+10);
  idelta_y = (int) (delta_y * 1.00e+10);
  idelta_t = (int) (delta_t * 1.00e+12);
  iforce_x = (int) (force_x * 1.00e+25);
  iforce_y = (int) (force_y * 1.00e+25);
  inzero1 = (int) (nzero1 * 100.000);
  inzero2 = (int) (nzero2 * 100.000);
  sprintf(id_name,
     "_%1d%1d%1d%1d_%3.3d_%3.3d_%3.3d_%4.4d_%5.5d_%3.3d_%4.4d_%5.5d_%5.5d_%2.2d_%9.9d_%9
.9d_%3.3d_%3.3d",
          key_scheme,key_init,key_boundary,key_interface,
          nnodes_x,nnodes_y,imass1,icspeed1,itau1,
          imass2,icspeed2,itau2,idelta_t,lambda,idelta_x,idelta_y,
          inzero1,inzero2);
          sprintf(rez_name,"R%s",id_name);
          sprintf(xv_name,"X%s",id_name);
}
```

```c
/***********************************************************************\
*                                                                       *
*  wet9aux.c                                                            *
*                                                                       *
\***********************************************************************/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>

#include "wet9head.h"

void init_lattice_functions(void)
{
  f10 = (double*) malloc(nnodes_all*sizeof(double));
  f11 = (double*) malloc(nnodes_all*sizeof(double));
  f12 = (double*) malloc(nnodes_all*sizeof(double));
  f13 = (double*) malloc(nnodes_all*sizeof(double));
  f14 = (double*) malloc(nnodes_all*sizeof(double));
  f15 = (double*) malloc(nnodes_all*sizeof(double));
  f16 = (double*) malloc(nnodes_all*sizeof(double));
  f17 = (double*) malloc(nnodes_all*sizeof(double));
  f18 = (double*) malloc(nnodes_all*sizeof(double));
  f20 = (double*) malloc(nnodes_all*sizeof(double));
  f21 = (double*) malloc(nnodes_all*sizeof(double));
  f22 = (double*) malloc(nnodes_all*sizeof(double));
  f23 = (double*) malloc(nnodes_all*sizeof(double));
  f24 = (double*) malloc(nnodes_all*sizeof(double));
  f25 = (double*) malloc(nnodes_all*sizeof(double));
  f26 = (double*) malloc(nnodes_all*sizeof(double));
  f27 = (double*) malloc(nnodes_all*sizeof(double));
  f28 = (double*) malloc(nnodes_all*sizeof(double));
  ff10 = (double*) malloc(nnodes_all*sizeof(double));
  ff11 = (double*) malloc(nnodes_all*sizeof(double));
  ff12 = (double*) malloc(nnodes_all*sizeof(double));
  ff13 = (double*) malloc(nnodes_all*sizeof(double));
  ff14 = (double*) malloc(nnodes_all*sizeof(double));
  ff15 = (double*) malloc(nnodes_all*sizeof(double));
  ff16 = (double*) malloc(nnodes_all*sizeof(double));
  ff17 = (double*) malloc(nnodes_all*sizeof(double));
  ff18 = (double*) malloc(nnodes_all*sizeof(double));
  ff20 = (double*) malloc(nnodes_all*sizeof(double));
  ff21 = (double*) malloc(nnodes_all*sizeof(double));
  ff22 = (double*) malloc(nnodes_all*sizeof(double));
  ff23 = (double*) malloc(nnodes_all*sizeof(double));
  ff24 = (double*) malloc(nnodes_all*sizeof(double));
  ff25 = (double*) malloc(nnodes_all*sizeof(double));
  ff26 = (double*) malloc(nnodes_all*sizeof(double));
  ff27 = (double*) malloc(nnodes_all*sizeof(double));
  ff28 = (double*) malloc(nnodes_all*sizeof(double));
  neq10 = (double*) malloc(nnodes_all*sizeof(double));
  neq11 = (double*) malloc(nnodes_all*sizeof(double));
  neq12 = (double*) malloc(nnodes_all*sizeof(double));
  neq13 = (double*) malloc(nnodes_all*sizeof(double));
  neq14 = (double*) malloc(nnodes_all*sizeof(double));
  neq15 = (double*) malloc(nnodes_all*sizeof(double));
  neq16 = (double*) malloc(nnodes_all*sizeof(double));
  neq17 = (double*) malloc(nnodes_all*sizeof(double));
  neq18 = (double*) malloc(nnodes_all*sizeof(double));
  neq20 = (double*) malloc(nnodes_all*sizeof(double));
  neq21 = (double*) malloc(nnodes_all*sizeof(double));
  neq22 = (double*) malloc(nnodes_all*sizeof(double));
  neq23 = (double*) malloc(nnodes_all*sizeof(double));
  neq24 = (double*) malloc(nnodes_all*sizeof(double));
  neq25 = (double*) malloc(nnodes_all*sizeof(double));
  neq26 = (double*) malloc(nnodes_all*sizeof(double));
  neq27 = (double*) malloc(nnodes_all*sizeof(double));
  neq28 = (double*) malloc(nnodes_all*sizeof(double));
  source10 = (double*) malloc(nnodes_all*sizeof(double));
```



```c
  source11 = (double*) malloc(nnodes_all*sizeof(double));
  source12 = (double*) malloc(nnodes_all*sizeof(double));
  source13 = (double*) malloc(nnodes_all*sizeof(double));
  source14 = (double*) malloc(nnodes_all*sizeof(double));
  source15 = (double*) malloc(nnodes_all*sizeof(double));
  source16 = (double*) malloc(nnodes_all*sizeof(double));
  source17 = (double*) malloc(nnodes_all*sizeof(double));
  source18 = (double*) malloc(nnodes_all*sizeof(double));
  source20 = (double*) malloc(nnodes_all*sizeof(double));
  source21 = (double*) malloc(nnodes_all*sizeof(double));
  source22 = (double*) malloc(nnodes_all*sizeof(double));
  source23 = (double*) malloc(nnodes_all*sizeof(double));
  source24 = (double*) malloc(nnodes_all*sizeof(double));
  source25 = (double*) malloc(nnodes_all*sizeof(double));
  source26 = (double*) malloc(nnodes_all*sizeof(double));
  source27 = (double*) malloc(nnodes_all*sizeof(double));
  source28 = (double*) malloc(nnodes_all*sizeof(double));
  sf10 = (double*) malloc(nnodes_all*sizeof(double));
  sf11 = (double*) malloc(nnodes_all*sizeof(double));
  sf12 = (double*) malloc(nnodes_all*sizeof(double));
  sf13 = (double*) malloc(nnodes_all*sizeof(double));
  sf14 = (double*) malloc(nnodes_all*sizeof(double));
  sf15 = (double*) malloc(nnodes_all*sizeof(double));
  sf16 = (double*) malloc(nnodes_all*sizeof(double));
  sf17 = (double*) malloc(nnodes_all*sizeof(double));
  sf18 = (double*) malloc(nnodes_all*sizeof(double));
  sf20 = (double*) malloc(nnodes_all*sizeof(double));
  sf21 = (double*) malloc(nnodes_all*sizeof(double));
  sf22 = (double*) malloc(nnodes_all*sizeof(double));
  sf23 = (double*) malloc(nnodes_all*sizeof(double));
  sf24 = (double*) malloc(nnodes_all*sizeof(double));
  sf25 = (double*) malloc(nnodes_all*sizeof(double));
  sf26 = (double*) malloc(nnodes_all*sizeof(double));
  sf27 = (double*) malloc(nnodes_all*sizeof(double));
  sf28 = (double*) malloc(nnodes_all*sizeof(double));
  uxloc = (double*) malloc(nnodes_all*sizeof(double));
  uyloc = (double*) malloc(nnodes_all*sizeof(double));
  uxloc1 = (double*) malloc(nnodes_all*sizeof(double));
  uyloc1 = (double*) malloc(nnodes_all*sizeof(double));
  uxloc2 = (double*) malloc(nnodes_all*sizeof(double));
  uyloc2 = (double*) malloc(nnodes_all*sizeof(double));
  nv1 = (int*) malloc(nnodes_all*sizeof(int));
  nv2 = (int*) malloc(nnodes_all*sizeof(int));
  nv3 = (int*) malloc(nnodes_all*sizeof(int));
  nv4 = (int*) malloc(nnodes_all*sizeof(int));
  nv5 = (int*) malloc(nnodes_all*sizeof(int));
  nv6 = (int*) malloc(nnodes_all*sizeof(int));
  nv7 = (int*) malloc(nnodes_all*sizeof(int));
  nv8 = (int*) malloc(nnodes_all*sizeof(int));
  boundary_mode = (int*) malloc(nnodes_all*sizeof(int));
  nloc1 = (double*) malloc(nnodes_all*sizeof(double));
  nloc2 = (double*) malloc(nnodes_all*sizeof(double));
  colorfield = (double*) malloc(nnodes_all*sizeof(double));
  gradcolor_x = (double*) malloc(nnodes_all*sizeof(double));
  gradcolor_y = (double*) malloc(nnodes_all*sizeof(double));
  gradn1x = (double*) malloc(nnodes_all*sizeof(double));
  gradn2x = (double*) malloc(nnodes_all*sizeof(double));
  gradn1y = (double*) malloc(nnodes_all*sizeof(double));
  gradn2y = (double*) malloc(nnodes_all*sizeof(double));
}

void free_lattice_functions(void)
{
  free(f10);
  free(f11);
  free(f12);
  free(f13);
  free(f14);
  free(f15);
```

```
  free(f16);
  free(f17);
  free(f18);
  free(f20);
  free(f21);
  free(f22);
  free(f23);
  free(f24);
  free(f25);
  free(f26);
  free(f27);
  free(f28);
  free(ff10);
  free(ff11);
  free(ff12);
  free(ff13);
  free(ff14);
  free(ff15);
  free(ff16);
  free(ff17);
  free(ff18);
  free(ff20);
  free(ff21);
  free(ff22);
  free(ff23);
  free(ff24);
  free(ff25);
  free(ff26);
  free(ff27);
  free(ff28);
  free(neq10);
  free(neq11);
  free(neq12);
  free(neq13);
  free(neq14);
  free(neq15);
  free(neq16);
  free(neq17);
  free(neq18);
  free(neq20);
  free(neq21);
  free(neq22);
  free(neq23);
  free(neq24);
  free(neq25);
  free(neq26);
  free(neq27);
  free(neq28);
  free(source10);
  free(source11);
  free(source12);
  free(source13);
  free(source14);
  free(source15);
  free(source16);
  free(source17);
  free(source18);
  free(source20);
  free(source21);
  free(source22);
  free(source23);
  free(source24);
  free(source25);
  free(source26);
  free(source27);
  free(source28);
  free(sf10);
  free(sf11);
  free(sf12);
```



```
  free(sf13);
  free(sf14);
  free(sf15);
  free(sf16);
  free(sf17);
  free(sf18);
  free(sf20);
  free(sf21);
  free(sf22);
  free(sf23);
  free(sf24);
  free(sf25);
  free(sf26);
  free(sf27);
  free(sf28);
  free(uxloc);
  free(uyloc);
  free(uxloc1);
  free(uyloc1);
  free(uxloc2);
  free(uyloc2);
  free(nv1);
  free(nv2);
  free(nv3);
  free(nv4);
  free(nv5);
  free(nv6);
  free(nv7);
  free(nv8);
  free(boundary_mode);
  free(nloc1);
  free(nloc2);
  free(colorfield);
  free(gradn1x);
  free(gradn2x);
  free(gradn1y);
  free(gradn2y);
}


void test_distribution_functions(double nf0[], double nf1[], double nf2[],
                                 double nf3[], double nf4[], double nf5[],
                                 double nf6[], double nf7[], double nf8[])
{
  FILE *frez;
  int k;
  frez = fopen(rez_name,"aw");
  for(k=0; k<nnodes_all; k++)
    {
      if(nf0[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf0=%g\n",
                  iter,k,nf0[k]);
          exit(1);
        }
      if(nf1[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf1=%g\n",
                  iter,k,nf1[k]);
          exit(1);
        }
      if(nf2[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf2=%g\n",
                  iter,k,nf2[k]);
          exit(1);
        }
      if(nf3[k] < 0.00)
        {
```

```
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf3=%g\n",
                  iter,k,nf3[k]);
          exit(1);
        }
      if(nf4[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf4=%g\n",
                  iter,k,nf4[k]);
          exit(1);
        }
      if(nf5[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf5=%g\n",
                  iter,k,nf5[k]);
          exit(1);
        }
      if(nf6[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf6=%g\n",
                  iter,k,nf6[k]);
          exit(1);
        }
      if(nf7[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf7=%g\n",
                  iter,k,nf7[k]);
          exit(1);
        }
      if(nf8[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf8=%g\n",
                  iter,k,nf8[k]);
          exit(1);
        }
    }
  fclose(frez);
}
```

```c
/*********************************************************************\
*                                                                     *
*  wet9init.c  --  arrays initialisation                              *
*                                                                     *
\*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "wet9head.h"

void print_tavec(int kk, int nv[])
{
  int i,j;
  FILE *fout;
  fout = fopen(rez_name,"aw");
  fprintf(fout,"\nnv%1d\n",kk);
  for(j=0;j<=nnodes_y;j++)
    {
      for(i=0;i<=nnodes_x;i++)
        {
          fprintf(fout,"%3d ",nv[(j-1)*nnodes_x+i]);
        }
      fprintf(fout,"\n");
    }
  fclose(fout);
}

void getavec_square(void)
{
  int i, j, k;
  k = 0;
  for(j=0; j<nnodes_y; j++)
    {
      for(i=0; i<nnodes_x; i++)
        {
          nv1[k] = k+1;
          if(i == (nnodes_x-1))
            nv1[k] -= nnodes_x;
          nv2[k] = k+nnodes_x;
          if(nv2[k] >= nnodes_all)
            nv2[k] -= nnodes_all;
          nv3[k] = k-1;
          if(i == 0)
            nv3[k] += nnodes_x;
          nv4[k] = k-nnodes_x;
          if(nv4[k] < 0)
            nv4[k]+= nnodes_all;
          nv5[k] = nv2[k] + 1;
          if(i == (nnodes_x-1))
            nv5[k] -= nnodes_x;
          nv6[k] = nv2[k] - 1;
          if(i == 0)
            nv6[k] += nnodes_x;
          nv7[k] = nv4[k] - 1;
          if(i == 0)
            nv7[k] += nnodes_x;
          nv8[k] = nv4[k] + 1;
          if(i == (nnodes_x-1))
            nv8[k] -= nnodes_x;
          /*
            printf("k=%d  %d  %d  %d  %d  %d  %d  %d  %d\n",
            k,nv1[k],nv2[k],nv3[k],nv4[k],nv5[k],nv6[k],nv7[k],nv8[k]);
          */
          k++;
        }
    }
```



```c
}

void init_ecx_nine(void)
{
  int i;

  double xminusminus, xminus, xzero, xplus, xx, xy;
  double lxminusminus, lxminus, lxzero, lxplus,
         lyminusminus, lyminus, lyzero, lyplus;
  double yminus, yzero, yplus;
  double xtl, xtr, xbl, xbr, ytl, ytr, ybl, ybr;

  ww[0] = w0;
  ww[1] = w1;
  ww[2] = w1;
  ww[3] = w1;
  ww[4] = w1;
  ww[5] = w2;
  ww[6] = w2;
  ww[7] = w2;
  ww[8] = w2;

  ecx[0] = ((double) 0);
  ecy[0] = ((double) 0);
  ecx[1] = ((double) 1);
  ecy[1] = ((double) 0);
  ecx[2] = ((double) 0);
  ecy[2] = ((double) 1);
  ecx[3] = ((double) -1);
  ecy[3] = ((double) 0);;
  ecx[4] = ((double) 0);
  ecy[4] = ((double) -1);
  ecx[5] = ((double) 1);
  ecy[5] = ((double) 1);
  ecx[6] = ((double) -1);;
  ecy[6] = ((double) 1);
  ecx[7] = ((double) -1);
  ecy[7] = ((double) -1);
  ecx[8] = ((double) 1);;
  ecy[8] = ((double) -1);

  for( i=0; i < 9; i++)
    {
      ecx1[i] = ecx[i] * cspeed1;
      ecy1[i] = ecy[i] * cspeed1;
      ecx2[i] = ecx[i] * cspeed2;
      ecy2[i] = ecy[i] * cspeed2;

      ecxforce1[i] = delta_t * force_x * ecx[i] * cspeed1 / (kboltz * temp);
      ecyforce1[i] = delta_t * force_y * ecy[i] * cspeed1 / (kboltz * temp);
      ecprod1[i] = ecxforce1[i] + ecyforce1[i];
      ecxforce2[i] = delta_t * force_x * ecx[i] * cspeed2 / (kboltz * temp);
      ecyforce2[i] = delta_t * force_y * ecy[i] * cspeed2 / (kboltz * temp);
      ecprod2[i] = ecxforce2[i] + ecyforce2[i];
    }

  xminus = -1.000;
  xzero = 0.000;
  xplus = 1.000;

  for(i=1; i<9; i++)
    {
      xx = xzero - ecx1[i] * delta_t / delta_x;
      xy = xzero - ecy1[i] * delta_t / delta_x;

      lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
      lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
      lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));
```

```
      lyminus = ((xy-xzero)*(xy-xplus))/((xminus-xzero)*(xminus-xplus));
      lyzero = ((xy-xminus)*(xy-xplus))/((xzero-xminus)*(xzero-xplus));
      lyplus = ((xy-xminus)*(xy-xzero))/((xplus-xminus)*(xplus-xzero));

      nc10[i] = lxzero * lyzero;
      nc11[i] = lxplus * lyzero;
      nc12[i] = lxzero * lyplus;
      nc13[i] = lxminus * lyzero;
      nc14[i] = lxzero * lyminus;
      nc15[i] = lxplus * lyplus;
      nc16[i] = lxminus * lyplus;
      nc17[i] = lxminus * lyminus;
      nc18[i] = lxplus * lyminus;

      xx = xzero - ecx2[i] * delta_t / delta_x;
      xy = xzero - ecy2[i] * delta_t / delta_x;

      lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
      lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
      lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

      lyminus = ((xy-xzero)*(xy-xplus))/((xminus-xzero)*(xminus-xplus));
      lyzero = ((xy-xminus)*(xy-xplus))/((xzero-xminus)*(xzero-xplus));
      lyplus = ((xy-xminus)*(xy-xzero))/((xplus-xminus)*(xplus-xzero));

      nc20[i] = lxzero * lyzero;
      nc21[i] = lxplus * lyzero;
      nc22[i] = lxzero * lyplus;
      nc23[i] = lxminus * lyzero;
      nc24[i] = lxzero * lyminus;
      nc25[i] = lxplus * lyplus;
      nc26[i] = lxminus * lyplus;
      nc27[i] = lxminus * lyminus;
      nc28[i] = lxplus * lyminus;
    }

  xminus = -1.000;
  xzero = 0.000;
  xplus = 1.000;
  yminus = -1.000;
  yzero = 0.000;
  yplus = 1.000;

  for(i=1; i<9; i++)
    {
      xx = xzero - ecx1[i] * delta_t / delta_x;
      xy = xzero - ecy1[i] * delta_t / delta_x;

      lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
      lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
      lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

      lyzero = (xy-yplus)/(yzero-yplus);
      lyplus = (xy-yzero)/(yplus-yzero);

      ncbot10[i] = lxzero * lyzero;
      ncbot11[i] = lxplus * lyzero;
      ncbot12[i] = lxzero * lyplus;
      ncbot13[i] = lxminus * lyzero;
      ncbot15[i] = lxplus * lyplus;
      ncbot16[i] = lxminus * lyplus;

      xx = xzero - ecx2[i] * delta_t / delta_x;
      xy = xzero - ecy2[i] * delta_t / delta_x;

      lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
      lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
      lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));
```



```
      lyzero = (xy-yplus)/(yzero-yplus);
      lyplus = (xy-yzero)/(yplus-yzero);

      ncbot20[i] = lxzero * lyzero;
      ncbot21[i] = lxplus * lyzero;
      ncbot22[i] = lxzero * lyplus;
      ncbot23[i] = lxminus * lyzero;
      ncbot25[i] = lxplus * lyplus;
      ncbot26[i] = lxminus * lyplus;

      xx = xzero - ecx1[i] * delta_t / delta_x;
      xy = xzero - ecy1[i] * delta_t / delta_x;

      lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
      lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
      lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

      lyzero = (xy-yminus)/(yzero-yminus);
      lyminus = (xy-yzero)/(yminus-yzero);

      nctop10[i] = lxzero * lyzero;
      nctop11[i] = lxplus * lyzero;
      nctop13[i] = lxminus * lyzero;
      nctop17[i] = lxminus * lyminus;
      nctop14[i] = lxzero * lyminus;
      nctop18[i] = lxplus * lyminus;

      xx = xzero - ecx2[i] * delta_t / delta_x;
      xy = xzero - ecy2[i] * delta_t / delta_x;

      lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
      lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
      lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

      lyzero = (xy-yminus)/(yzero-yminus);
      lyminus = (xy-yzero)/(yminus-yzero);

      nctop20[i] = lxzero * lyzero;
      nctop21[i] = lxplus * lyzero;
      nctop23[i] = lxminus * lyzero;
      nctop27[i] = lxminus * lyminus;
      nctop24[i] = lxzero * lyminus;
      nctop28[i] = lxplus * lyminus;

      xx = xzero - ecx1[i] * delta_t / delta_x;
      xy = xzero - ecy1[i] * delta_t / delta_x;

      lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
      lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
      lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

      lxzero = (xx-xplus)/(xzero-xplus);
      lxplus = (xx-xzero)/(xplus-xzero);

      ncleft15[i] = lxplus * lyplus;
      ncleft11[i] = lxplus * lyzero;
      ncleft18[i] = lxplus * lyminus;
      ncleft12[i] = lxzero * lyplus;
      ncleft10[i] = lxzero * lyzero;
      ncleft14[i] = lxzero * lyminus;

      xx = xzero - ecx2[i] * delta_t / delta_x;
      xy = xzero - ecy2[i] * delta_t / delta_x;

      lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
      lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
      lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

      lxzero = (xx-xplus)/(xzero-xplus);
```

```
        lxplus = (xx-xzero)/(xplus-xzero);

        ncleft25[i] = lxplus * lyplus;
        ncleft21[i] = lxplus * lyzero;
        ncleft28[i] = lxplus * lyminus;
        ncleft22[i] = lxzero * lyplus;
        ncleft20[i] = lxzero * lyzero;
        ncleft24[i] = lxzero * lyminus;

        xx = xzero - ecx1[i] * delta_t / delta_x;
        xy = xzero - ecy1[i] * delta_t / delta_x;

        lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
        lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
        lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

        lxzero = (xx-xminus)/(xzero-xminus);
        lxminus = (xx-xzero)/(xminus-xzero);

        ncright10[i] = lxzero * lyzero;
        ncright12[i] = lxzero * lyplus;
        ncright14[i] = lxzero * lyminus;
        ncright16[i] = lxminus * lyplus;
        ncright13[i] = lxminus * lyzero;
        ncright17[i] = lxminus * lyminus;

        xx = xzero - ecx2[i] * delta_t / delta_x;
        xy = xzero - ecy2[i] * delta_t / delta_x;

        lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
        lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
        lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

        lxzero = (xx-xminus)/(xzero-xminus);
        lxminus = (xx-xzero)/(xminus-xzero);

        ncright20[i] = lxzero * lyzero;
        ncright22[i] = lxzero * lyplus;
        ncright24[i] = lxzero * lyminus;
        ncright26[i] = lxminus * lyplus;
        ncright23[i] = lxminus * lyzero;
        ncright27[i] = lxminus * lyminus;

    }

  xminus = -1.000;
  xzero = 0.000;
  xplus = 1.000;

  yminus = -1.000;
  yzero = 0.000;
  yplus = 1.000;

  i=1;
  xx = xplus - ecx1[i] * delta_t / delta_x;
  xy = yzero - ecy1[i] * delta_t / delta_x;

  lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
  lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
  lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

  lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
  lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
  lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

  nu10[i] = lxzero  * lyzero;
  nu11[i] = lxplus  * lyzero;
  nu12[i] = lxzero  * lyplus;
  nu13[i] = lxminus * lyzero;
```



```
  nu14[i] = lxzero  * lyminus;
  nu15[i] = lxplus  * lyplus;
  nu16[i] = lxminus * lyplus;
  nu17[i] = lxminus * lyminus;
  nu18[i] = lxplus  * lyminus;

  i=2;
  xx = xzero - ecx1[i] * delta_t / delta_x;
  xy = yplus - ecy1[i] * delta_t / delta_x;

  lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
  lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
  lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

  lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
  lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
  lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

  nu10[i] = lxzero  * lyzero;
  nu11[i] = lxplus  * lyzero;
  nu12[i] = lxzero  * lyplus;
  nu13[i] = lxminus * lyzero;
  nu14[i] = lxzero  * lyminus;
  nu15[i] = lxplus  * lyplus;
  nu16[i] = lxminus * lyplus;
  nu17[i] = lxminus * lyminus;
  nu18[i] = lxplus  * lyminus;

  i=3;
  xx = xminus - ecx1[i] * delta_t / delta_x;
  xy = yzero - ecy1[i]  * delta_t / delta_x;

  lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
  lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
  lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

  lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
  lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
  lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

  nu10[i] = lxzero  * lyzero;
  nu11[i] = lxplus  * lyzero;
  nu12[i] = lxzero  * lyplus;
  nu13[i] = lxminus * lyzero;
  nu14[i] = lxzero  * lyminus;
  nu15[i] = lxplus  * lyplus;
  nu16[i] = lxminus * lyplus;
  nu17[i] = lxminus * lyminus;
  nu18[i] = lxplus  * lyminus;

  i=4;
  xx = xzero - ecx1[i]  * delta_t / delta_x;
  xy = yminus - ecy1[i] * delta_t / delta_x;

  lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
  lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
  lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

  lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
  lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
  lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

  nu10[i] = lxzero  * lyzero;
  nu11[i] = lxplus  * lyzero;
  nu12[i] = lxzero  * lyplus;
  nu13[i] = lxminus * lyzero;
  nu14[i] = lxzero  * lyminus;
  nu15[i] = lxplus  * lyplus;
  nu16[i] = lxminus * lyplus;
```

```
nu17[i] = lxminus * lyminus;
nu18[i] = lxplus  * lyminus;

i=5;
xx = xplus - ecx1[i] * delta_t / delta_x;
xy = yplus - ecy1[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu10[i] = lxzero  * lyzero;
nu11[i] = lxplus  * lyzero;
nu12[i] = lxzero  * lyplus;
nu13[i] = lxminus * lyzero;
nu14[i] = lxzero  * lyminus;
nu15[i] = lxplus  * lyplus;
nu16[i] = lxminus * lyplus;
nu17[i] = lxminus * lyminus;
nu18[i] = lxplus  * lyminus;

i=6;
xx = xminus - ecx1[i] * delta_t / delta_x;
xy = yplus - ecy1[i]  * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu10[i] = lxzero  * lyzero;
nu11[i] = lxplus  * lyzero;
nu12[i] = lxzero  * lyplus;
nu13[i] = lxminus * lyzero;
nu14[i] = lxzero  * lyminus;
nu15[i] = lxplus  * lyplus;
nu16[i] = lxminus * lyplus;
nu17[i] = lxminus * lyminus;
nu18[i] = lxplus  * lyminus;

i=7;
xx = xminus - ecx1[i] * delta_t / delta_x;
xy = yminus - ecy1[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu10[i] = lxzero  * lyzero;
nu11[i] = lxplus  * lyzero;
nu12[i] = lxzero  * lyplus;
nu13[i] = lxminus * lyzero;
nu14[i] = lxzero  * lyminus;
nu15[i] = lxplus  * lyplus;
nu16[i] = lxminus * lyplus;
nu17[i] = lxminus * lyminus;
nu18[i] = lxplus  * lyminus;
```



```
i=8;
xx = xplus - ecx1[i]  * delta_t / delta_x;
xy = yminus - ecy1[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu10[i] = lxzero  * lyzero;
nu11[i] = lxplus  * lyzero;
nu12[i] = lxzero  * lyplus;
nu13[i] = lxminus * lyzero;
nu14[i] = lxzero  * lyminus;
nu15[i] = lxplus  * lyplus;
nu16[i] = lxminus * lyplus;
nu17[i] = lxminus * lyminus;
nu18[i] = lxplus  * lyminus;

i=1;
xx = xplus - ecx2[i] * delta_t / delta_x;
xy = yzero - ecy2[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

i=2;
xx = xzero - ecx2[i] * delta_t / delta_x;
xy = yplus - ecy2[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

i=3;
xx = xminus - ecx2[i] * delta_t / delta_x;
xy = yzero -  ecy2[i] * delta_t / delta_x;
```

```
lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

i=4;
xx = xzero -  ecx2[i] * delta_t / delta_x;
xy = yminus - ecy2[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

i=5;
xx = xplus - ecx2[i] * delta_t / delta_x;
xy = yplus - ecy2[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

i=6;
xx = xminus - ecx2[i] * delta_t / delta_x;
xy = yplus -  ecy2[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
```



```
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

i=7;
xx = xminus - ecx2[i] * delta_t / delta_x;
xy = yminus - ecy2[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

i=8;
xx = xplus -  ecx2[i] * delta_t / delta_x;
xy = yminus - ecy2[i] * delta_t / delta_x;

lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus));
lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus));
lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));

lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus));
lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus));
lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));

nu20[i] = lxzero  * lyzero;
nu21[i] = lxplus  * lyzero;
nu22[i] = lxzero  * lyplus;
nu23[i] = lxminus * lyzero;
nu24[i] = lxzero  * lyminus;
nu25[i] = lxplus  * lyplus;
nu26[i] = lxminus * lyplus;
nu27[i] = lxminus * lyminus;
nu28[i] = lxplus  * lyminus;

/*
xminusminus = -1.000;
xminus = 0.000;
xzero = 1.000;

for(i=1; i<9; i++)
  {
    xx = xzero - ecx1[i] * delta_t / delta_x;
    xy = xzero - ecy1[i] * delta_t / delta_x;
```

```
    lxminusminus = ((xx-xminus)*(xx-xzero))/
                   ((xminusminus-xminus)*(xminusminus-xzero));
    lxminus = ((xx-xminusminus)*(xx-xzero))/
              ((xminus-xminusminus)*(xminus-xzero));
    lxzero = ((xx-xminusminus)*(xx-xminus))/
             ((xzero-xminusminus)*(xzero-xminus));

    lyminusminus = ((xy-xminus)*(xy-xzero))/
                   ((xminusminus-xminus)*(xminusminus-xzero));
    lyminus = ((xy-xminusminus)*(xy-xzero))/
              ((xminus-xminusminus)*(xminus-xzero));
    lyzero = ((xy-xminusminus)*(xy-xminus))/
             ((xzero-xminusminus)*(xzero-xminus));

    nu10[i] = lxminus * lyminus;
    nu11[i] = lxzero * lyminus;
    nu12[i] = lxminus * lyzero;
    nu13[i] = lxminusminus * lyminus;
    nu14[i] = lxminus * lyminusminus;
    nu15[i] = lxzero * lyzero;
    nu16[i] = lxminusminus * lyzero;
    nu17[i] = lxminusminus * lyminusminus;
    nu18[i] = lxzero * lyminusminus;

    xx = xzero - ecx2[i] * delta_t / delta_x;
    xy = xzero - ecy2[i] * delta_t / delta_x;

    lxminusminus = ((xx-xminus)*(xx-xzero))/
                   ((xminusminus-xminus)*(xminusminus-xzero));
    lxminus = ((xx-xminusminus)*(xx-xzero))/
              ((xminus-xminusminus)*(xminus-xzero));
    lxzero = ((xx-xminusminus)*(xx-xminus))/
             ((xzero-xminusminus)*(xzero-xminus));

    lyminusminus = ((xy-xminus)*(xy-xzero))/
                   ((xminusminus-xminus)*(xminusminus-xzero));
    lyminus = ((xy-xminusminus)*(xy-xzero))/
              ((xminus-xminusminus)*(xminus-xzero));
    lyzero = ((xy-xminusminus)*(xy-xminus))/
             ((xzero-xminusminus)*(xzero-xminus));

    nu20[i] = lxminus * lyminus;
    nu21[i] = lxzero * lyminus;
    nu22[i] = lxminus * lyzero;
    nu23[i] = lxminusminus * lyminus;
    nu24[i] = lxminus * lyminusminus;
    nu25[i] = lxzero * lyzero;
    nu26[i] = lxminusminus * lyzero;
    nu27[i] = lxminusminus * lyminusminus;
    nu28[i] = lxzero * lyminusminus;
  }
*/
/* serendip */

xtl = -1.000;
xtr =  1.000;
xbl = -1.000;
xbr =  1.000;

ytl =  1.000;
ytr =  1.000;
ybl = -1.000;
ybr = -1.000;

for(i=1; i<9; i++)
  {
    switch(i)
      {
```



```
        case 1:
        case 5:
          xx = xtr - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ytr - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        case 2:
        case 6:
          xx = xtl - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ytl - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        case 3:
        case 7:
          xx = xbl - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ybl - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        case 4:
        case 8:
          xx = xbr - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ybr - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
      }
    /*
    printf("i1=%d  xx=%e  xy=%e\n",i,xx,xy);
    */

    ns1tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
    ns1tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
    ns1bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
    ns1br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

    /*
    printf("tl=%e  tr=%e  bl=%e  br=%e\n",
           ns1tl[i],ns1tr[i],ns1bl[i],ns1br[i]);
    */
    nl1tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
    nl1tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
    nl1br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
    nl1bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

    switch(i)
      {
        case 1:
        case 5:
          xx = xtr - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ytr - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        case 2:
        case 6:
          xx = xtl - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ytl - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        case 3:
        case 7:
          xx = xbl - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ybl - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        case 4:
        case 8:
          xx = xbr - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ybr - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
      }
    /*
    printf("i2=%d  xx=%e  xy=%e\n",i,xx,xy);
    */
    ns2tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
    ns2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
    ns2bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
    ns2br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;
```

```
      /*
      printf("tl=%e  tr=%e  bl=%e  br=%e\n",
             ns2tl[i],ns2tr[i],ns2bl[i],ns2br[i]);
      */
      nl2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nl2tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nl2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nl2bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

    }

  for(i=1; i<9; i++)
    {
      switch(i)
        {
        case 2:
        case 5:
          xx = xbl - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ybl - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        case 1:
        case 6:
          xx = xbr - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ybr - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        default:
          break;
        }

      nsb1tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nsb1tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nsb1bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nsb1br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

      nlb1tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nlb1tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nlb1br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nlb1bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

      switch(i)
        {
        case 3:
        case 8:
          xx = xtl - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ytl - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        case 4:
        case 7:
          xx = xtr - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ytr - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        default:
          break;
        }

      nst1tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nst1tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nst1bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nst1br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

      nlt1tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nlt1tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nlt1br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nlt1bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

      switch(i)
        {
        case 1:
        case 8:
```



```
          xx = xtl - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ytl - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        case 4:
        case 5:
          xx = xbl - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ybl - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        default:
          break;
        }

      nsl1tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nsl1tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nsl1bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nsl1br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

      nll1tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nll1tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nll1br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nll1bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

      switch(i)
        {
        case 2:
        case 7:
          xx = xtr - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ytr - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        case 3:
        case 6:
          xx = xbr - 2.000 * ecx1[i] * delta_t / delta_x;
          xy = ybr - 2.000 * ecy1[i] * delta_t / delta_x;
          break;
        default:
          break;
        }

      nsr1tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nsr1tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nsr1bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nsr1br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

      nlr1tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nlr1tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nlr1br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nlr1bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

      switch(i)
        {
        case 2:
        case 5:
          xx = xbl - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ybl - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        case 1:
        case 6:
          xx = xbr - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ybr - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        default:
          break;
        }

      nsb2tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nsb2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nsb2bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nsb2br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;
```

```
      nlb2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nlb2tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nlb2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nlb2bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

      switch(i)
        {
        case 3:
        case 8:
          xx = xtl - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ytl - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        case 4:
        case 7:
          xx = xtr - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ytr - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        default:
          break;
        }

      nst2tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nst2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nst2bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nst2br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

      nlt2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nlt2tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nlt2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nlt2bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

      switch(i)
        {
        case 1:
        case 8:
          xx = xtl - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ytl - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        case 4:
        case 5:
          xx = xbl - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ybl - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        default:
          break;
        }

      nsl2tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nsl2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nsl2bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nsl2br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

      nll2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nll2tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nll2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nll2bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

      switch(i)
        {
        case 2:
        case 7:
          xx = xtr - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ytr - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
        case 3:
        case 6:
          xx = xbr - 2.000 * ecx2[i] * delta_t / delta_x;
          xy = ybr - 2.000 * ecy2[i] * delta_t / delta_x;
          break;
```



```
        default:
          break;
        }

      nsr2tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;
      nsr2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000;
      nsr2bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;
      nsr2br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;

      nlr2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr));
      nlr2tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl));
      nlr2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr));
      nlr2bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));

    }
}

void init_arrays_nine_square(void)
{
  FILE *fsav, *frez;
  int i,j,k, key_rho;
  double rand_coef = 0.000000000001,  dummy;
  double radius, xc, yc, x,y;

  frez=fopen(rez_name,"a");

  for( i=0; i<9; i++)
    {
      fprintf(frez,"i,ecx[i],ecy[i]\n%3d %lf %lf\n",
              i,ecx[i],ecy[i]);
      fprintf(frez,"i,ecx1[i],ecy1[i]\n%3d %lf %lf\n",
              i,ecx1[i],ecy1[i]);
      fprintf(frez,"i,ecx2[i],ecy2[i]\n%3d %lf %lf\n",
              i,ecx2[i],ecy2[i]);
    }


  switch (key_init)
    {
    case 0: /*  homogeneous binary system  */
      k=-1;
      for ( j=0; j< nnodes_y; j++)
        {
          for( i=0; i < nnodes_x; i++)
            {
              k++;
              f10[k] = nzero1 * (1.000 + rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
              f20[k] = nzero2 * (1.000 + rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
            }
        }
      break;
    case 1: /*  two horizontal layers  */
      k=-1;
      for ( j=0; j< nnodes_y; j++)
        {
          for( i=0; i < nnodes_x; i++)
            {
              k++;
              if(j<nnodes_y/2)
                {
                  f10[k] = nzero1 * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = 0.000e+0;
                }
              else
                {
                  f10[k] = 0.000e+0;
```

```
              f10[k] = nzero2 * (1.000 + rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
            }
          }
      }
      break;
    case 2: /*  diffusion couple  */
      k=-1;
      for ( j=0; j< nnodes_y; j++)
        {
          for( i=0; i < nnodes_x; i++)
            {
              k++;
              if(i<=nnodes_x/2)
                {
                  f10[k] = nzero1left * (1. +rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = nzero2left * (1. +rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                }
              else
                {
                  f10[k] = nzero1right * (1.0 +rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = nzero2right * (1.0 +rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                }
            }
        }
      break;
    case 3: /*  circular drop  */
      radius = delta_x * ((double) nnodes_y) / 4.000;
      xc = delta_x * ((double) nnodes_x) / 2.000;
      yc = delta_x * ((double) nnodes_y) / 2.000;
      nzero1 = nzero1left;
      nzero2 = nzero2left;
      k = -1;
      for(j=0; j<nnodes_y; j++)
        {
          y = delta_x * ((double) j);
          for(i=0; i<nnodes_x; i++)
            {
              x = delta_x * ((double) i);
              k++;
              if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=
                 (radius+3.0*delta_x)*(radius+3.0*delta_x))
                 /*
                 radius*radius)
                 */
                key_rho = 4;
              else
                key_rho = 5;
              if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=
                 (radius+2.0*delta_x)*(radius+2.0*delta_x))
                key_rho = 3;
              if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=
                 (radius+delta_x)*(radius+delta_x))
                key_rho = 2;
              if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=radius*radius)
                key_rho = 1;
              switch(key_rho)
                {
                case 1:
                  f10[k] = nzero1left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = nzero2right * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  break;
                case 2:
```



```
                  f10[k] = nzero1left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5))/4.00;
                  f20[k] = nzero2left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5))/4.00;
                  break;
                case 3:
                  f10[k] = nzero1left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5))/2.00;
                  f20[k] = nzero2left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5))/2.00;
                  break;
                case 4:
                  f10[k] = 3.000 * nzero1left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5))/4.00;
                  f20[k] = 3.000 * nzero2left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5))/4.00;
                  break;
                case 5:
                  f10[k] = nzero1right * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = nzero2left * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  break;
                }
            }
        }
      break;
    case 4: /*  circular bubble  */
      break;
    case 5: /*  sessile drop  */
      break;
    case 6: /*  pendant drop  */
      break;
    case 7: /*  sessile bubble  */
      break;
    case 8: /*  pendant bubble  */
      break;
    case 9: /*  liquid bridge  */
      k=-1;
      for ( j=0; j< nnodes_y; j++)
        {
          for( i=0; i < nnodes_x; i++)
            {
              k++;
              if( (i<nnodes_x/4) || (i>3*nnodes_x/4) )
                {
                  f10[k] = 0.000e+0;
                  f20[k] = nzero2 * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                }
              else
                {
                  f10[k] = nzero1 * (1.000 + rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = 0.000e+0;
                }
            }
        }
      break;
    case 10: /*  gas bridge  */
      k=-1;
      for ( j=0; j< nnodes_y; j++)
        {
          for( i=0; i < nnodes_x; i++)
            {
              k++;
              if( (i<nnodes_x/3) || (i>2*nnodes_x/3) )
                {
```

```c
              f10[k] = nzero1 * (1.000 + rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
              f20[k] = 0.000e+0;
            }
          else
            {
              f10[k] = 0.000e+0;
              f20[k] = nzero2 * (1.000 + rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
            }
        }
      }
    break;
   }

  for( k=0; k < nnodes_all; k++)
    {
      f11[k] = w1 * f10[k];
      f12[k] = w1 * f10[k];
      f13[k] = w1 * f10[k];
      f14[k] = w1 * f10[k];
      f15[k] = w2 * f10[k];
      f16[k] = w2 * f10[k];
      f17[k] = w2 * f10[k];
      f18[k] = w2 * f10[k];
      f10[k] *= w0;
      f21[k] = w1 * f20[k];
      f22[k] = w1 * f20[k];
      f23[k] = w1 * f20[k];
      f24[k] = w1 * f20[k];
      f25[k] = w2 * f20[k];
      f26[k] = w2 * f20[k];
      f27[k] = w2 * f20[k];
      f28[k] = w2 * f20[k];
      f20[k] *= w0;
    }

  fclose(frez);
}

void init_arrays_nine_square_aux(void)
{
  int i, j, nnl1 = (nnodes_y-1)*nnodes_x;
  switch(key_boundary)
    {
    case 0:  /*   periodic boundaries  */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 0;
      break;
    case 1:  /*   horizontal walls placed top and bottom   */
      for ( i = 0; i< nnodes_x; i++)
        boundary_mode[i] = 1;  /*  bottom wall  */
      for ( i = nnodes_x; i < nnodes_all-nnodes_x; i++)
        boundary_mode[i] = 0;  /*  bulk  */
      for ( i = nnodes_all-nnodes_x; i < nnodes_all; i++)
        boundary_mode[i] = 2;  /*  top wall  */
      if(key_scheme == 6)
        {
          for(i=nnodes_x; i<2*nnodes_x; i++)
            boundary_mode[i] = 11;
          for(i=nnodes_all-(2*nnodes_x); i<nnodes_all-nnodes_x; i++)
            boundary_mode[i] = 22;
        }
      break;
    case 2:  /*   vertical walls placed left and right   */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 0;  /*  bulk  */
      for ( j = 0; j < nnodes_y; j++)
        {
```



```c
          boundary_mode[j*nnodes_x] = 3;  /*  left wall  */
          boundary_mode[(j+1)*nnodes_x-1] = 4;  /*  right wall  */
        }
      break;
    }
}
```

```c
/*************************************************************\
*                                                             *
*  wet9draw.c                                                 *
*                                                             *
\*************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "wet9head.h"

/* you may modify gray_levels */
  int gray_levels = 63, base = 35;

void xv_new(double n0[],
            double n1[], double n2[], double n3[], double n4[],
            double n5[], double n6[], double n7[], double n8[],
            char arg_name[], double maxval)
{
  int i, val;
  char xv_name[128];
  FILE *fxv;

  sprintf(xv_name, "%s.%06d.xpm", arg_name, iter);
  fxv = fopen(xv_name, "w");
  fprintf(fxv, "/* XPM */\nstatic char * init_xpm[] = {\n");
  fprintf(fxv, "\"%d %d %d %d", nnodes_x, nnodes_y, gray_levels + 1, 1);
  for(i = 0; i <= gray_levels; i++)
    fprintf(fxv, "\",\n\"%c\tg gray%d", base + i,
            (int)(100.00 * i / gray_levels));

                                        /* lattice -> xpm */
  for(i = 0; i <= nnodes_all; i++)
    {
      if(1 == ((i+1) % nnodes_x))
        fprintf(fxv,"\",\n\"");
      val = floor( (n0[i] + n1[i] + n2[i] + n3[i] + n4[i] + n5[i] +
                  n6[i] + n7[i] + n8[i]) * gray_levels / maxval);
      if(val > gray_levels)
        val = gray_levels;
      if(val < 0)
        val = 0;
      fprintf(fxv,"%c", base + val);
    }
  fprintf(fxv,"\"};\n");
  fclose(fxv);
}


void xv(double n0[],
        double n1[], double n2[], double n3[], double n4[],
        double n5[], double n6[], double n7[], double n8[],
        char arg_name[])
{
  FILE *fxv;
  char xv_name[128];
  int i,k,val;
  sprintf(xv_name,"%s.%05d",arg_name,iter);
  fxv = fopen(xv_name,"wt");
  fprintf(fxv,"P2\n%3d%4d\n63\n",nnodes_x,nnodes_y);
  i=0;
  for(k=0; k<nnodes_all; k++)
    {
      i++;
      val = floor((n0[k]+n1[k]+n2[k]+n3[k]+n4[k]+n5[k]+
                  n6[k]+n7[k]+n8[k])*63.0);
      if(val > 63)
```



```c
        val = 63;
      if(val < 0)
        val = 0;
/*      val = 63 - val;   */
      fprintf(fxv,"%3d",val);
      if(i == 15)
        {
          i=0;
          fprintf(fxv,"\n");
        }
    }
  if(i)
    fprintf(fxv,"\n");
  fclose(fxv);
}

void wet9_drop_profile(void)
{
  FILE *pn1x, *pn2x;

  char n1x_profile_name[128];
  char n2x_profile_name[128];

  int i,j,k;
  double nloc1, nloc2;
  double n1x[nnodes_x], n2x[nnodes_x];

  sprintf(n1x_profile_name,"PRESS1X%s.%06d",id_name,iter);
  sprintf(n2x_profile_name,"PRESS2X%s.%06d",id_name,iter);

  pn1x = fopen(n1x_profile_name,"wt");
  pn2x = fopen(n2x_profile_name,"wt");

  k = nnodes_x * nnodes_y/2 - 1;
  for(i=0; i<nnodes_x; i++)
    {
      k ++;
      nloc1 = f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+
        f17[k]+f18[k];
      nloc2 = f20[k]+f21[k]+f22[k]+f23[k]+f24[k]+f25[k]+f26[k]+
        f27[k]+f28[k];

      fprintf(pn1x,"%d  %25.20lf\n",i,nloc1);
      fprintf(pn2x,"%d  %25.20lf\n",i,nloc2);
    }

  fclose(pn1x);
  fclose(pn2x);
}

void wet9_channel_profile(void)
{
  FILE *pn1, *pn2, *pntot, *prhotot, *pvx, *pvx1, *pvx2, *pgvx, *pvisco;

  FILE *pf0, *pf1, *pf2, *pf3, *pf4, *pf5, *pf6, *pf7, *pf8;

  char n1_profile_name[128];
  char n2_profile_name[128];
  char ntot_profile_name[128];
  char rhotot_profile_name[128];
  char vx_profile_name[128];
  char grad_vx_profile_name[128];
  char vx1_profile_name[128];
  char vx2_profile_name[128];
  char visco_name[128];

  char f0_name[128];
  char f1_name[128];
  char f2_name[128];
```

```
char f3_name[128];
char f4_name[128];
char f5_name[128];
char f6_name[128];
char f7_name[128];
char f8_name[128];

int i,j,k;
double nloc1, nloc2, nloc, rho1, rho2, rholoc, uloc1, uloc2, uloc, mean_mass;
double vx[nnodes_y], y[nnodes_y], grad_vx[nnodes_y];
double vx1[nnodes_y], vx2[nnodes_y];
double n1[nnodes_y], n2[nnodes_y], ntot[nnodes_y], rhotot[nnodes_y];
double visco, ivisco, sum1, sum2, channel_width;

sprintf(n1_profile_name,"PN1%s.%06d",id_name,iter);
sprintf(n2_profile_name,"PN2%s.%06d",id_name,iter);
sprintf(ntot_profile_name,"PNTOT%s.%06d",id_name,iter);
sprintf(rhotot_profile_name,"PRHOTOT%s.%06d",id_name,iter);
sprintf(vx_profile_name,"PVX%s.%06d",id_name,iter);
sprintf(vx1_profile_name,"PVX1%s.%06d",id_name,iter);
sprintf(vx2_profile_name,"PVX2%s.%06d",id_name,iter);
sprintf(grad_vx_profile_name,"PGVX%s.%06d",id_name,iter);
sprintf(visco_name,"PVISCO%s",id_name);

sprintf(f0_name,"PF0%s.%06d",id_name,iter);
sprintf(f1_name,"PF1%s.%06d",id_name,iter);
sprintf(f2_name,"PF2%s.%06d",id_name,iter);
sprintf(f3_name,"PF3%s.%06d",id_name,iter);
sprintf(f4_name,"PF4%s.%06d",id_name,iter);
sprintf(f5_name,"PF5%s.%06d",id_name,iter);
sprintf(f6_name,"PF6%s.%06d",id_name,iter);
sprintf(f7_name,"PF7%s.%06d",id_name,iter);
sprintf(f8_name,"PF8%s.%06d",id_name,iter);

pf0 = fopen(f0_name,"wt");
pf1 = fopen(f1_name,"wt");
pf2 = fopen(f2_name,"wt");
pf3 = fopen(f3_name,"wt");
pf4 = fopen(f4_name,"wt");
pf5 = fopen(f5_name,"wt");
pf6 = fopen(f6_name,"wt");
pf7 = fopen(f7_name,"wt");
pf8 = fopen(f8_name,"wt");

pn1 = fopen(n1_profile_name,"wt");
pn2 = fopen(n2_profile_name,"wt");
pntot = fopen(ntot_profile_name,"wt");
prhotot = fopen(rhotot_profile_name,"wt");
pvx = fopen(vx_profile_name,"wt");
pvx1 = fopen(vx1_profile_name,"wt");
pvx2 = fopen(vx2_profile_name,"wt");
pgvx = fopen(grad_vx_profile_name,"wt");
if(iter)
  pvisco = fopen(visco_name,"awt");
else
  pvisco = fopen(visco_name,"wt");

k = 0;
for(j=0; j<nnodes_y; j++)
  {
    y[j] = ((double) j) * delta_y;
    vx[j] = 0.0000;
    vx1[j] = 0.0000;
    vx2[j] = 0.0000;
    n1[j] = 0.0000;
    n2[j] = 0.0000;
    ntot[j] = 0.0000;
    rhotot[j] = 0.0000;
    for(i=0; i<nnodes_x; i++)
```



```
      {
       nloc1 = f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+
               f17[k]+f18[k];
       nloc2 = f20[k]+f21[k]+f22[k]+f23[k]+f24[k]+f25[k]+f26[k]+
               f27[k]+f28[k];
       nloc = nloc1 + nloc2;
       rho1 = nloc1 * mass1;
       rho2 = nloc2 * mass2;
       rholoc = rho1 + rho2;
       switch(boundary_mode[k])
         {
         case 0:
           if(nloc1)
             uloc1 = cspeed1*
               (f11[k]*ecx[1]+f12[k]*ecx[2]+f13[k]*ecx[3]+
                f14[k]*ecx[4]+f15[k]*ecx[5]+f16[k]*ecx[6]+
                f17[k]*ecx[7]+f18[k]*ecx[8]) / nloc1;
           else
             uloc1 = 0.0000;
           if(nloc2)
             uloc2 = cspeed2*
               (f21[k]*ecx[1]+f22[k]*ecx[2]+f23[k]*ecx[3]+
                f24[k]*ecx[4]+f25[k]*ecx[5]+f26[k]*ecx[6]+
                f27[k]*ecx[7]+f28[k]*ecx[8]) / nloc2;
           else
             uloc2 = 0.0000;
           if(rholoc)
             uloc = (rho1*uloc1 + rho2*uloc2) / rholoc;
           else
             uloc = 0.0000;
           break;
         case 1:
           uloc = uloc1 = uloc2 = uxwall_bot;
           break;
         case 2:
           uloc = uloc1 = uloc2 = uxwall_top;
           break;
         }
       n1[j] += nloc1;
       n2[j] += nloc2;
       ntot[j] += nloc;
       rhotot[j] += rholoc;
       vx[j] += uloc;
       vx1[j] += uloc1;
       vx2[j] += uloc2;
       k++;
      }
    n1[j] /= ((double) nnodes_x);
    n2[j] /= ((double) nnodes_x);
    ntot[j] /= ((double) nnodes_x);
    rhotot[j] /= ((double) nnodes_x);
    vx[j] /= ((double) nnodes_x);
    vx1[j] /= ((double) nnodes_x);
    vx2[j] /= ((double) nnodes_x);
    /*
      printf("draw j u1 u2 u: %d  %lf  %lf  %lf\n",j,vx[j],vx1[j],vx2[j]);
    */
    fprintf(pn1,"%lf  %25.20lf\n",y[j],n1[j]);
    fprintf(pn2,"%lf  %25.20lf\n",y[j],n2[j]);
    fprintf(pntot,"%lf  %25.20lf\n",y[j],ntot[j]);
    fprintf(prhotot,"%lf  %25.20lf\n",y[j],rhotot[j]);
    fprintf(pvx,"%lf  %25.20lf\n",y[j],vx[j]);
    fprintf(pvx1,"%lf  %25.20lf\n",y[j],vx1[j]);
    fprintf(pvx2,"%lf  %25.20lf\n",y[j],vx2[j]);

    fprintf(pf0,"%lf  %25.20lf\n",y[j],f10[j*nnodes_x]);
    fprintf(pf1,"%lf  %25.20lf\n",y[j],f11[j*nnodes_x]);
    fprintf(pf2,"%lf  %25.20lf\n",y[j],f12[j*nnodes_x]);
    fprintf(pf3,"%lf  %25.20lf\n",y[j],f13[j*nnodes_x]);
```

```
      fprintf(pf4,"%lf  %25.20lf\n",y[j],f14[j*nnodes_x]);
      fprintf(pf5,"%lf  %25.20lf\n",y[j],f15[j*nnodes_x]);
      fprintf(pf6,"%lf  %25.20lf\n",y[j],f16[j*nnodes_x]);
      fprintf(pf7,"%lf  %25.20lf\n",y[j],f17[j*nnodes_x]);
      fprintf(pf8,"%lf  %25.20lf\n",y[j],f18[j*nnodes_x]);
    }

  grad_vx[0] = (vx[1]-vx[0])/delta_y;
  fprintf(pgvx,"%lf  %25.20lf\n",y[0],grad_vx[0]);
  for(j=1; j<nnodes_y-1; j++)
    {
      grad_vx[j] = (vx[j+1]-vx[j-1])/(2.0000*delta_y);
      fprintf(pgvx,"%lf  %25.20lf\n",y[j],grad_vx[j]);
    }
  grad_vx[nnodes_y-1] = (vx[nnodes_y-1]-vx[nnodes_y-2])/delta_y;
  fprintf(pgvx,"%lf  %25.20lf\n",y[nnodes_y-1],grad_vx[nnodes_y-1]);

  channel_width = ((double) (nnodes_y-1)) * delta_y;
  sum1 = sum2 = 0.00000;
  for(j=0; j<nnodes_y; j++)
    {
      sum1 += (y[j]*channel_width - y[j]*y[j]) *
        (y[j]*channel_width - y[j]*y[j]);
      sum2 += vx[j] * (y[j]*channel_width - y[j]*y[j]);
    }
  sum2 *= 2.0000;
  mean_mass = (mass1*nzero1 + mass2*nzero2) / (nzero1+nzero2);
  /*
  if(vx[nnodes_y/2])
    printf("%lf  %lf  %e  %e  %e\n",mean_mass,(force_x*channel_width*channel_width)/
           (8.000*mean_mass*amu*vx[nnodes_y/2]),kboltz,temp,amu);
  else
    printf("%lf\n",mean_mass);
  */
  if(sum2)
    visco = (force_x / (mean_mass * amu)) * sum1 / sum2;
  else
    visco = 0.0000;
  fprintf(pvisco,"%8.8d   %e\n",iter,visco);

  fclose(pn1);
  fclose(pn2);
  fclose(pntot);
  fclose(prhotot);
  fclose(pvx1);
  fclose(pvx2);
  fclose(pvx);
  fclose(pgvx);
  fclose(pvisco);

  fclose(pf0);
  fclose(pf1);
  fclose(pf2);
  fclose(pf3);
  fclose(pf4);
  fclose(pf5);
  fclose(pf6);
  fclose(pf7);
  fclose(pf8);
}

void wet9_couple_profile(void)
{
  FILE *pntot, *prhotot, *prho1, *prho2, *pomega1, *pomegan1,
    *pomega2, *pomegan2, *pj2, *pu1, *pu2, *pu, *pu2bis, *pubis;

  char ntot_profile_name[128];
  char rhotot_profile_name[128];
  char rho1_profile_name[128];
```



```
  char rho2_profile_name[128];
  char omega1_profile_name[128];
  char omegan1_profile_name[128];
  char omega2_profile_name[128];
  char omegan2_profile_name[128];
  char j2_profile_name[128];
  char u1_profile_name[128];
  char u2_profile_name[128];
  char u_profile_name[128];
  char u2bis_profile_name[128];
  char ubis_profile_name[128];

  int i,j,k,kk,nval;
  double nloc1, nloc2, rholoc1, rholoc2, rholoc, omega1loc, omegan1loc,
    omega2loc, omegan2loc, uloc1, uloc2, jloc2, uloc;
  double uloc1bis, uloc2bis, ulocbis;
  double rho1, rho2, rho, omega1, omegan1, omega2, omegan2,
    u1, u2, j2, u, u2bis, ubis;
  double ntot, rhotot;

  sprintf(ntot_profile_name,"PNTOTAL%s.%06d",id_name,iter);
  sprintf(rhotot_profile_name,"PRHOTOT%s.%06d",id_name,iter);
  sprintf(rho1_profile_name,"PRHO1%s.%06d",id_name,iter);
  sprintf(rho2_profile_name,"PRHO2%s.%06d",id_name,iter);
  sprintf(omega1_profile_name,"POMEGAMASS1%s.%06d",id_name,iter);
  sprintf(omegan1_profile_name,"POMEGAN1%s.%06d",id_name,iter);
  sprintf(omega2_profile_name,"POMEGAMASS2%s.%06d",id_name,iter);
  sprintf(omegan2_profile_name,"POMEGAN2%s.%06d",id_name,iter);
  sprintf(j2_profile_name,"PJ2%s.%06d",id_name,iter);
  sprintf(u1_profile_name,"PU1%s.%06d",id_name,iter);
  sprintf(u2_profile_name,"PU2%s.%06d",id_name,iter);
  sprintf(u_profile_name,"PU%s.%06d",id_name,iter);
  sprintf(u2bis_profile_name,"PU2BIS%s.%06d",id_name,iter);
  sprintf(ubis_profile_name,"PUBIS%s.%06d",id_name,iter);

  pntot = fopen(ntot_profile_name,"wt");
  prhotot = fopen(rhotot_profile_name,"wt");
  prho1 = fopen(rho1_profile_name,"wt");
  prho2 = fopen(rho2_profile_name,"wt");
  pomega1 = fopen(omega1_profile_name,"wt");
  pomegan1 = fopen(omegan1_profile_name,"wt");
  pomega2 = fopen(omega2_profile_name,"wt");
  pomegan2 = fopen(omegan2_profile_name,"wt");
  pj2 = fopen(j2_profile_name,"wt");
  pu1 = fopen(u1_profile_name,"wt");
  pu2 = fopen(u2_profile_name,"wt");
  pu = fopen(u_profile_name,"wt");
  /*
  pu2bis = fopen(u2bis_profile_name,"wt");
  pubis = fopen(ubis_profile_name,"wt");
  */
  for(i=0; i<nnodes_x; i++)
    {
      ntot = 0.0000;
      rhotot = 0.0000;
      rho1 = 0.0000;
      rho2 = 0.0000;
      omega1 = 0.0000;
      omegan1 = 0.0000;
      omega2 = 0.0000;
      omegan2 = 0.0000;
      j2 = 0.0000;
      u1 = 0.0000;
      u2 = 0.0000;
      u = 0.0000;
      u2bis = 0.0000;
      ubis = 0.0000;
      for(j=0; j<nnodes_y; j++)
        {
```

```c
        k = i + j * nnodes_x;
        nloc1 = f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+
                f17[k]+f18[k];
        nloc2 = f20[k]+f21[k]+f22[k]+f23[k]+f24[k]+f25[k]+f26[k]+
                f27[k]+f28[k];
        /*
        printf("k=%d  nloc1=%g  nloc2=%g\n",k,nloc1,nloc2);
        */
        rholoc1 = mass1 * nloc1;
        rholoc2 = mass2 * nloc2;
        rholoc = rholoc1 + rholoc2;
        /*
        printf("k=%d  rholoc1=%g  rholoc2=%g  rholoc=%g\n",
               k,rholoc1,rholoc2,rholoc);
        */
        if(rholoc)
          {
          omega1loc = rholoc1 / rholoc;
          omega2loc = rholoc2 / rholoc;
          }
        else
          {
            omega1loc = 0.0000;
            omega2loc = 0.0000;
            /*
            printf("k=%d  rholoc=%g\n",k,rholoc);
            */
          }

        if(nloc1+nloc2)
          {
          omegan1loc = nloc1 / (nloc1+nloc2);
          omegan2loc = nloc2 / (nloc1+nloc2);
          }
        else
          {
            omegan1loc = 0.0000;
            omegan2loc = 0.0000;
          }

        if(nloc1)
          uloc1 = (f11[k]*ecx1[1]+f12[k]*ecx1[2]+f13[k]*ecx1[3]+
                   f14[k]*ecx1[4]+f15[k]*ecx1[5]+f16[k]*ecx1[6]+
                   f17[k]*ecx1[7]+f18[k]*ecx1[8]) / nloc1;
        else
          uloc1 = 0.0000;

        if(nloc2)
          uloc2 = (f21[k]*ecx2[1]+f22[k]*ecx2[2]+f23[k]*ecx2[3]+
                   f24[k]*ecx2[4]+f25[k]*ecx2[5]+f26[k]*ecx2[6]+
                   f27[k]*ecx2[7]+f28[k]*ecx2[8]) / nloc2;
        else
          uloc2 = 0.0000;

        /*
        printf("k=%d  uloc1=%g  uloc2=%g\n",k,uloc1,uloc2);
        */

        if(rholoc)
          {
            uloc = ( rholoc1*uloc1 + rholoc2*uloc2 ) / rholoc;
            jloc2 = rholoc2 * (uloc2 - uloc);
          }
        else
          {
            uloc = 0.0000;
            /*
            printf("k=%d  rholoc=%g\n",k,rholoc);
            */
```



```c
          }

        /*
        if(nloc1)
          uloc1bis = (uloc1 * (1.000 - 0.500*delta_t/tau1) +
                      uloc*0.500*delta_t/tau1)/nloc1;
        else
          uloc1bis = 0.000;
        if(nloc2)
          uloc2bis = (uloc2 * (1.000 - 0.500*delta_t/tau2) +
                      uloc*0.500*delta_t/tau2)/nloc2;
        else
          uloc2bis = 0.000;
        ulocbis = (rholoc1*uloc1*(1.000 - 0.500*delta_t/tau1) +
                   rholoc2*uloc2*(1.000 - 0.500*delta_t/tau2) +
                   rholoc1*uloc*0.500*delta_t/tau1 +
                   rholoc2*uloc*0.500*delta_t/tau2) / rholoc;
        */

        ntot += (nloc1 + nloc2);
        rhotot += (rholoc1 + rholoc2);
        rho1 += rholoc1;
        rho2 += rholoc2;
        omega1 += omega1loc;
        omega2 += omega2loc;
        omegan1 += omegan1loc;
        omegan2 += omegan2loc;
        j2 += jloc2;
        u1 += uloc1;
        u2 += uloc2;
        u += uloc;
        /*
        u2bis += uloc2bis;
        ubis += ulocbis;
        */
      }
    ntot /= ((double) nnodes_y);
    rhotot /= ((double) nnodes_y);
    rho1 /= ((double) nnodes_y);
    rho2 /= ((double) nnodes_y);
    omega1 /= ((double) nnodes_y);
    omega2 /= ((double) nnodes_y);
    omegan1 /= ((double) nnodes_y);
    omegan2 /= ((double) nnodes_y);
    j2 /= ((double) nnodes_y);
    u1 /= ((double) nnodes_y);
    u2 /= ((double) nnodes_y);
    u /= ((double) nnodes_y);
    u2bis /= ((double) nnodes_y);
    ubis /= ((double) nnodes_y);
    fprintf(pntot,"%g\n",ntot);
    fprintf(prhotot,"%g\n",rhotot);
    fprintf(prho1,"%g\n",rho1);
    fprintf(prho2,"%g\n",rho2);
    fprintf(pomega1,"%lf\n",omega1);
    fprintf(pomegan1,"%lf\n",omegan1);
    fprintf(pomega2,"%lf\n",omega2);
    fprintf(pomegan2,"%lf\n",omegan2);
    fprintf(pj2,"%lf\n",j2);
    fprintf(pu1,"%lf\n",u1);
    fprintf(pu2,"%lf\n",u2);
    fprintf(pu,"%lf\n",u);
    /*
    fprintf(pu2bis,"%lf\n",u2bis);
    fprintf(pubis,"%lf\n",ubis);
    */
  }

  fclose(pntot);
```

```
  fclose(prhotot);
  fclose(prho1);
  fclose(prho2);
  fclose(pomega1);
  fclose(pomegan1);
  fclose(pomega2);
  fclose(pomegan2);
  fclose(pj2);
  fclose(pu1);
  fclose(pu2);
  fclose(pu);
  /*
  fclose(pu2bis);
  fclose(pubis);
  */
}


void test_ntot(void)
{
  FILE *ptot;
  int k;
  double ntot1=0.0000, ntot2=0.0000;
  char ntot_name[128];
  sprintf(ntot_name,"PNTOT%s",id_name);
  ptot = fopen(ntot_name,"aw");
  for(k=0; k<nnodes_all; k++)
    {
      ntot1 += f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+f17[k]+f18[k];
      ntot2 += f20[k]+f21[k]+f22[k]+f23[k]+f24[k]+f25[k]+f26[k]+f27[k]+f28[k];
    }
  fprintf(ptot,"iter=%d  ntot1= %25.20lf  ntot2= %25.20lf \n",
          iter,ntot1,ntot2);
  fclose(ptot);
}

void quiver(void)
{
  FILE *pux, *puy, *pxc, *pyc, *pxn, *pyn;
  double nloc1, nloc2;
  double ux1, uy1, ux2, uy2, ux, uy, xc, yc;
  int i,j,k,ipas=2,jpas=2, xn, yn;
  char ux_name[128], uy_name[128];
  char cx_name[128], cy_name[128];
  char nx_name[128], ny_name[128];
  sprintf(ux_name,"PUX%s.%06d",id_name,iter);
  sprintf(uy_name,"PUY%s.%06d",id_name,iter);
  sprintf(cx_name,"PCX%s.%06d",id_name,iter);
  sprintf(cy_name,"PCY%s.%06d",id_name,iter);
  sprintf(nx_name,"PNX%s.%06d",id_name,iter);
  sprintf(ny_name,"PNY%s.%06d",id_name,iter);
  pux = fopen(ux_name,"w");
  puy = fopen(uy_name,"w");
  pxc = fopen(cx_name,"w");
  pyc = fopen(cy_name,"w");
  pxn = fopen(nx_name,"w");
  pyn = fopen(ny_name,"w");
  for(j=0; j<nnodes_y; j+=jpas)
    {
      k = j*nnodes_x;
      for(i=0; i<nnodes_x; i+=ipas)
        {
          nloc1 = f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k];
          nloc2 = f20[k]+f21[k]+f22[k]+f23[k]+f24[k]+f25[k]+f26[k];
          if(nloc1)
            {
              ux1 = (f11[k]*ecx[1]+f12[k]*ecx[2]+f13[k]*ecx[3]+
                     f14[k]*ecx[4]+f15[k]*ecx[5]+f16[k]*ecx[6]) / nloc1;
              uy1 = (f11[k]*ecy[1]+f12[k]*ecy[2]+f13[k]*ecy[3]+
```



```
                     f14[k]*ecy[4]+f15[k]*ecy[5]+f16[k]*ecy[6]) / nloc1;
            }
          else
            {
              ux1 = 0.0000;
              uy1 = 0.0000;
            }
          if(nloc2)
            {
              ux2 = (f21[k]*ecx[1]+f22[k]*ecx[2]+f23[k]*ecx[3]+
                     f24[k]*ecx[4]+f25[k]*ecx[5]+f26[k]*ecx[6]) / nloc2;
              uy2 = (f21[k]*ecy[1]+f22[k]*ecy[2]+f23[k]*ecy[3]+
                     f24[k]*ecy[4]+f25[k]*ecy[5]+f26[k]*ecy[6]) / nloc2;
            }
          else
            {
              ux2 = 0.0000;
              uy2 = 0.0000;
            }
          ux = (mass1 * nloc1 * ux1 + mass2 * nloc2 * ux2) /
               (mass1 * nloc1 + mass2 * nloc2);
          uy = (mass1 * nloc1 * uy1 + mass2 * nloc2 * uy2) /
               (mass1 * nloc1 + mass2 * nloc2);
          xc = delta_x * ((double) i);
          yc = delta_y * ((double) j);
          xn = i;
          yn = j;
          fprintf(pux,"  %lf",ux);
          fprintf(puy,"  %lf",uy);
          fprintf(pxc,"  %lf",xc);
          fprintf(pyc,"  %lf",yc);
          fprintf(pxn,"  %5d",xn);
          fprintf(pyn,"  %5d",yn);
          k+= ipas;
        }
      fprintf(pux,"\n");
      fprintf(puy,"\n");
      fprintf(pxc,"\n");
      fprintf(pyc,"\n");
      fprintf(pxn,"\n");
      fprintf(pyn,"\n");
    }
  fclose(pux);
  fclose(puy);
  fclose(pxc);
  fclose(pyc);
  fclose(pxn);
  fclose(pyn);
}
```

```c
/*******************************************************************\
*                                                                   *
*   wet9cs.c                                                        *
*                                                                   *
\*******************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "wet9head.h"

double compute_centered_sources_bulk(int sigma, int index, int k,
                              int nvplus, int nvminus,
                              double nf10[], double nf11[], double nf12[],
                              double nf13[], double nf14[], double nf15[],
                              double nf16[], double nf17[], double nf18[],
                              double nf20[], double nf21[], double nf22[],
                              double nf23[], double nf24[], double nf25[],
                              double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, cfl, source;
  double fs1[9], fs2[9];

  switch(sigma)
    {
    case 1:
      cfl = cfl1;
      break;
    case 2:
      cfl = cfl2;
      break;
    }

  fs1[0] = ( nf10[nvplus] + nf10[nvminus] ) / 2.000 -
     cfl * ( nf10[nvplus] - nf10[nvminus] );
  fs1[1] = ( nf11[nvplus] + nf11[nvminus] ) / 2.000 -
     cfl * ( nf11[nvplus] - nf11[nvminus] );
  fs1[2] = ( nf12[nvplus] + nf12[nvminus] ) / 2.000 -
     cfl * ( nf12[nvplus] - nf12[nvminus] );
  fs1[3] = ( nf13[nvplus] + nf13[nvminus] ) / 2.000 -
     cfl * ( nf13[nvplus] - nf13[nvminus] );
  fs1[4] = ( nf14[nvplus] + nf14[nvminus] ) / 2.000 -
     cfl * ( nf14[nvplus] - nf14[nvminus] );
  fs1[5] = ( nf15[nvplus] + nf15[nvminus] ) / 2.000 -
     cfl * ( nf15[nvplus] - nf15[nvminus] );
  fs1[6] = ( nf16[nvplus] + nf16[nvminus] ) / 2.000 -
     cfl * ( nf16[nvplus] - nf16[nvminus] );
  fs1[7] = ( nf17[nvplus] + nf17[nvminus] ) / 2.000 -
     cfl * ( nf17[nvplus] - nf17[nvminus] );
  fs1[8] = ( nf18[nvplus] + nf18[nvminus] ) / 2.000 -
     cfl * ( nf18[nvplus] - nf18[nvminus] );

  fs2[0] = ( nf20[nvplus] + nf20[nvminus] ) / 2.000 -
     cfl * ( nf20[nvplus] - nf20[nvminus] );
  fs2[1] = ( nf21[nvplus] + nf21[nvminus] ) / 2.000 -
     cfl * ( nf21[nvplus] - nf21[nvminus] );
  fs2[2] = ( nf22[nvplus] + nf22[nvminus] ) / 2.000 -
     cfl * ( nf22[nvplus] - nf22[nvminus] );
  fs2[3] = ( nf23[nvplus] + nf23[nvminus] ) / 2.000 -
     cfl * ( nf23[nvplus] - nf23[nvminus] );
  fs2[4] = ( nf24[nvplus] + nf24[nvminus] ) / 2.000 -
     cfl * ( nf24[nvplus] - nf24[nvminus] );
  fs2[5] = ( nf25[nvplus] + nf25[nvminus] ) / 2.000 -
     cfl * ( nf25[nvplus] - nf25[nvminus] );
  fs2[6] = ( nf26[nvplus] + nf26[nvminus] ) / 2.000 -
     cfl * ( nf26[nvplus] - nf26[nvminus] );
  fs2[7] = ( nf27[nvplus] + nf27[nvminus] ) / 2.000 -
     cfl * ( nf27[nvplus] - nf27[nvminus] );
```



```c
  fs2[8] = ( nf28[nvplus] + nf28[nvminus] ) / 2.000 -
     cfl * ( nf28[nvplus] - nf28[nvminus] );

  dummy = mass1 * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+
                   fs1[6]+fs1[7]+fs1[8]) / tau1 +
          mass2 * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+
                   fs2[6]+fs2[7]+fs2[8]) / tau2;
  ux = (mass1 * (fs1[1]*ecx1[1]+fs1[2]*ecx1[2]+fs1[3]*ecx1[3]+fs1[4]*ecx1[4]+
                 fs1[5]*ecx1[5]+fs1[6]*ecx1[6]+fs1[7]*ecx1[7]+fs1[8]*ecx1[8])
        /tau1 +
        mass2 * (fs2[1]*ecx2[1]+fs2[2]*ecx2[2]+fs2[3]*ecx2[3]+fs2[4]*ecx2[4]+
                 fs2[5]*ecx2[5]+fs2[6]*ecx2[6]+fs2[7]*ecx2[7]+fs2[8]*ecx2[8])
        /tau2) / dummy;
  uy = (mass1 * (fs1[1]*ecy1[1]+fs1[2]*ecy1[2]+fs1[3]*ecy1[3]+fs1[4]*ecy1[4]+
                 fs1[5]*ecy1[5]+fs1[6]*ecy1[6]+fs1[7]*ecy1[7]+fs1[8]*ecy1[8])
        /tau1 +
        mass2 * (fs2[1]*ecy2[1]+fs2[2]*ecy2[2]+fs2[3]*ecy2[3]+fs2[4]*ecy2[4]+
                 fs2[5]*ecy2[5]+fs2[6]*ecy2[6]+fs2[7]*ecy2[7]+fs2[8]*ecy2[8])
        /tau2) / dummy;

  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      /*
      printf("dummy=%25.20e\n",dummy);
      */
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                  fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
      /*
      printf("k=%d  index=%d  fs1=%25.20e  neq=%25.20e  source=%25.20e  fs1-neq=%25.20e
 fs1+source=%25.20e  ux=%e  uy=%e  ecprod=%e\n",
             k,index,fs1[index],neq,source,fs1[index]-neq,fs1[index]+source,ux,uy,ecprod
1[index]);
      */
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                  fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    }
  return source;
}

double compute_centered_sources_boundary1(int sigma, int index, int k,
                              int nvplus, int nvminus,
                              double ux_boundary, double uy_boundary,
                              double nf10[], double nf11[], double nf12[],
                              double nf13[], double nf14[], double nf15[],
                              double nf16[], double nf17[], double nf18[],
                              double nf20[], double nf21[], double nf22[],
                              double nf23[], double nf24[], double nf25[],
                              double nf26[], double nf27[], double nf28[])
{
  double dummy, uu, neq, cfl, source;
  double fs1[9], fs2[9];

  switch(sigma)
    {
```

```
    case 1:
      cfl = cfl1;
  fs1[0] = ( nf10[nvplus] + nf10[nvminus] ) / 2.000 -
      cfl * ( nf10[nvplus] - nf10[nvminus] );
  fs1[1] = ( nf11[nvplus] + nf11[nvminus] ) / 2.000 -
      cfl * ( nf11[nvplus] - nf11[nvminus] );
  fs1[2] = ( nf12[nvplus] + nf12[nvminus] ) / 2.000 -
      cfl * ( nf12[nvplus] - nf12[nvminus] );
  fs1[3] = ( nf13[nvplus] + nf13[nvminus] ) / 2.000 -
      cfl * ( nf13[nvplus] - nf13[nvminus] );
  fs1[4] = ( nf14[nvplus] + nf14[nvminus] ) / 2.000 -
      cfl * ( nf14[nvplus] - nf14[nvminus] );
  fs1[5] = ( nf15[nvplus] + nf15[nvminus] ) / 2.000 -
      cfl * ( nf15[nvplus] - nf15[nvminus] );
  fs1[6] = ( nf16[nvplus] + nf16[nvminus] ) / 2.000 -
      cfl * ( nf16[nvplus] - nf16[nvminus] );
  fs1[7] = ( nf17[nvplus] + nf17[nvminus] ) / 2.000 -
      cfl * ( nf17[nvplus] - nf17[nvminus] );
  fs1[8] = ( nf18[nvplus] + nf18[nvminus] ) / 2.000 -
      cfl * ( nf18[nvplus] - nf18[nvminus] );
        break;
    case 2:
      cfl = cfl2;
  fs2[0] = ( nf20[nvplus] + nf20[nvminus] ) / 2.000 -
      cfl * ( nf20[nvplus] - nf20[nvminus] );
  fs2[1] = ( nf21[nvplus] + nf21[nvminus] ) / 2.000 -
      cfl * ( nf21[nvplus] - nf21[nvminus] );
  fs2[2] = ( nf22[nvplus] + nf22[nvminus] ) / 2.000 -
      cfl * ( nf22[nvplus] - nf22[nvminus] );
  fs2[3] = ( nf23[nvplus] + nf23[nvminus] ) / 2.000 -
      cfl * ( nf23[nvplus] - nf23[nvminus] );
  fs2[4] = ( nf24[nvplus] + nf24[nvminus] ) / 2.000 -
      cfl * ( nf24[nvplus] - nf24[nvminus] );
  fs2[5] = ( nf25[nvplus] + nf25[nvminus] ) / 2.000 -
      cfl * ( nf25[nvplus] - nf25[nvminus] );
  fs2[6] = ( nf26[nvplus] + nf26[nvminus] ) / 2.000 -
      cfl * ( nf26[nvplus] - nf26[nvminus] );
  fs2[7] = ( nf27[nvplus] + nf27[nvminus] ) / 2.000 -
      cfl * ( nf27[nvplus] - nf27[nvminus] );
  fs2[8] = ( nf28[nvplus] + nf28[nvminus] ) / 2.000 -
      cfl * ( nf28[nvplus] - nf28[nvminus] );
        break;
    }

  uu = ux_boundary*ux_boundary + uy_boundary*uy_boundary;

  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux_boundary + ecy1[index]*uy_boundary) / cspeed12;
      /*
      printf("dummy=%25.20e\n",dummy);
      */
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                  fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x*ux_boundary - csforce_y*uy_boundary) * neq;
      break;
    case 2:
      dummy = (ecx2[index]*ux_boundary + ecy2[index]*uy_boundary) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                  fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x*ux_boundary - csforce_y*uy_boundary) * neq;
        break;
```



```
      }
    return source;
}

double compute_centered_sources_boundary2(int sigma, int index, int k, int nv,
                                double ux_boundary, double uy_boundary,
                                double nf10[], double nf11[], double nf12[],
                                double nf13[], double nf14[], double nf15[],
                                double nf16[], double nf17[], double nf18[],
                                double nf20[], double nf21[], double nf22[],
                                double nf23[], double nf24[], double nf25[],
                                double nf26[], double nf27[], double nf28[])
{
  double dummy, uu, neq, cfl, source;
  double fs1[9], fs2[9];

  switch(sigma)
    {
    case 1:
      fs1[0] = nf10[k] - cfl1 * (nf10[k] - nf10[nv]);
      fs1[1] = nf11[k] - cfl1 * (nf11[k] - nf11[nv]);
      fs1[2] = nf12[k] - cfl1 * (nf12[k] - nf12[nv]);
      fs1[3] = nf13[k] - cfl1 * (nf13[k] - nf13[nv]);
      fs1[4] = nf14[k] - cfl1 * (nf14[k] - nf14[nv]);
      fs1[5] = nf15[k] - cfl1 * (nf15[k] - nf15[nv]);
      fs1[6] = nf16[k] - cfl1 * (nf16[k] - nf16[nv]);
      fs1[7] = nf17[k] - cfl1 * (nf17[k] - nf17[nv]);
      fs1[8] = nf18[k] - cfl1 * (nf18[k] - nf18[nv]);
      break;
    case 2:
      fs2[0] = nf20[k] - cfl2 * (nf20[k] - nf20[nv]);
      fs2[1] = nf21[k] - cfl2 * (nf21[k] - nf21[nv]);
      fs2[2] = nf22[k] - cfl2 * (nf22[k] - nf22[nv]);
      fs2[3] = nf23[k] - cfl2 * (nf23[k] - nf23[nv]);
      fs2[4] = nf24[k] - cfl2 * (nf24[k] - nf24[nv]);
      fs2[5] = nf25[k] - cfl2 * (nf25[k] - nf25[nv]);
      fs2[6] = nf26[k] - cfl2 * (nf26[k] - nf26[nv]);
      fs2[7] = nf27[k] - cfl2 * (nf27[k] - nf27[nv]);
      fs2[8] = nf28[k] - cfl2 * (nf28[k] - nf28[nv]);
      break;
    }


  uu = ux_boundary*ux_boundary + uy_boundary*uy_boundary;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux_boundary + ecy1[index]*uy_boundary) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                  fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x*ux_boundary - csforce_y*uy_boundary) * neq;
      break;
    case 2:
      dummy = (ecx2[index] * ux_boundary + ecy2[index]*uy_boundary) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                  fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x*ux_boundary - csforce_y*uy_boundary) * neq;
        break;
    }
  return source;
}
```

```
void compute_centered_sources(double nf10[], double nf11[], double nf12[],
                              double nf13[], double nf14[], double nf15[],
                              double nf16[], double nf17[], double nf18[],
                              double nf20[], double nf21[], double nf22[],
                              double nf23[], double nf24[], double nf25[],
                              double nf26[], double nf27[], double nf28[])
{
  int k;

  for(k=0; k<nnodes_all; k++)
    {
      source10[k] =  - ctau1 * (nf10[k] - neq10[k]) +
        (ecprod1[0] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq10[k];
      source11[k] =  - ctau1 * (nf11[k] - neq11[k]) +
        (ecprod1[1] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq11[k];
      source12[k] =  - ctau1 * (nf12[k] - neq12[k]) +
        (ecprod1[2] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq12[k];
      source13[k] =  - ctau1 * (nf13[k] - neq13[k]) +
        (ecprod1[3] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq13[k];
      source14[k] =  - ctau1 * (nf14[k] - neq14[k]) +
        (ecprod1[4] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq14[k];
      source15[k] =  - ctau1 * (nf15[k] - neq15[k]) +
        (ecprod1[5] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq15[k];
      source16[k] =  - ctau1 * (nf16[k] - neq16[k]) +
        (ecprod1[6] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq16[k];
      source17[k] =  - ctau1 * (nf17[k] - neq17[k]) +
        (ecprod1[7] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq17[k];
      source18[k] =  - ctau1 * (nf18[k] - neq18[k]) +
        (ecprod1[8] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq18[k];

      source20[k] =  - ctau2 * (nf20[k] - neq20[k]) +
        (ecprod2[0] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq20[k];
      source21[k] =  - ctau2 * (nf21[k] - neq21[k]) +
        (ecprod2[1] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq21[k];
      source22[k] =  - ctau2 * (nf22[k] - neq22[k]) +
        (ecprod2[2] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq22[k];
      source23[k] =  - ctau2 * (nf23[k] - neq23[k]) +
        (ecprod2[3] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq23[k];
      source24[k] =  - ctau2 * (nf24[k] - neq24[k]) +
        (ecprod2[4] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq24[k];
      source25[k] =  - ctau2 * (nf25[k] - neq25[k]) +
        (ecprod2[5] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq25[k];
      source26[k] =  - ctau2 * (nf26[k] - neq26[k]) +
        (ecprod2[6] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq26[k];
      source27[k] =  - ctau2 * (nf27[k] - neq27[k]) +
        (ecprod2[7] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq27[k];
      source28[k] =  - ctau2 * (nf28[k] - neq28[k]) +
        (ecprod2[8] - csforce_x * uxloc[k] - csforce_y * uyloc[k]) * neq28[k];
    }
}

void centered_cfl(double cfl,
                  double nf0[], double nf1[], double nf2[],
                  double nf3[], double nf4[], double nf5[],
                  double nf6[], double nf7[], double nf8[],
                  double nff0[], double nff1[], double nff2[],
                  double nff3[], double nff4[], double nff5[],
                  double nff6[], double nff7[], double nff8[],
                  double source0[], double source1[], double source2[],
                  double source3[], double source4[], double source5[],
                  double source6[], double source7[], double source8[])
{
  int k;

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
```



```
        case 0:
          nff0[k] = nf0[k] + source0[k];
          nff1[k] = (nf1[nv1[k]]+nf1[nv3[k]])/2.000 -
            cfl * (nf1[nv1[k]]-nf1[nv3[k]])/2.000 + source1[k];
          nff2[k] = (nf2[nv2[k]]+nf2[nv4[k]])/2.000 -
            cfl * (nf2[nv2[k]]-nf2[nv4[k]])/2.000 + source2[k];
          nff3[k] = (nf3[nv3[k]]+nf3[nv1[k]])/2.000 -
            cfl * (nf3[nv3[k]]-nf3[nv1[k]])/2.000 + source3[k];
          nff4[k] = (nf4[nv4[k]]+nf4[nv2[k]])/2.000 -
            cfl * (nf4[nv4[k]]-nf4[nv2[k]])/2.000 + source4[k];
          nff5[k] = (nf5[nv5[k]]+nf5[nv7[k]])/2.000 -
            cfl * (nf5[nv5[k]]-nf5[nv7[k]])/2.000 + source5[k];
          nff6[k] = (nf6[nv6[k]]+nf6[nv8[k]])/2.000 -
            cfl * (nf6[nv6[k]]-nf6[nv8[k]])/2.000 + source6[k];
          nff7[k] = (nf7[nv7[k]]+nf7[nv5[k]])/2.000 -
            cfl * (nf7[nv7[k]]-nf7[nv5[k]])/2.000 + source7[k];
          nff8[k] = (nf8[nv8[k]]+nf8[nv6[k]])/2.000 -
            cfl * (nf8[nv8[k]]-nf8[nv6[k]])/2.000 + source8[k];
          break;
        case 1:
          nff0[k] = nf0[k] + source0[k];
          nff1[k] = (nf1[nv1[k]]+nf1[nv3[k]])/2.000 -
            cfl * (nf1[nv1[k]]-nf1[nv3[k]])/2.000 + source1[k];
          nff2[k] = nf2[k] - cfl * (nf2[nv2[k]]-nf2[k]) + source2[k];
          nff3[k] = (nf3[nv3[k]]+nf3[nv1[k]])/2.000 -
            cfl * (nf3[nv3[k]]-nf3[nv1[k]])/2.000 + source3[k];
          nff4[k] = nf4[k] + cfl * (nf4[nv2[k]]-nf4[k]) + source4[k];
          nff5[k] = nf5[k] - cfl * (nf5[nv5[k]]-nf5[k]) + source5[k];
          nff6[k] = nf6[k] - cfl * (nf6[nv6[k]]-nf6[k]) + source6[k];
          nff7[k] = nf7[k] + cfl * (nf7[nv5[k]]-nf7[k]) + source7[k];
          nff8[k] = nf8[k] + cfl * (nf8[nv6[k]]-nf8[k]) + source8[k];
          break;
        case 2:
          nff0[k] = nf0[k] + source0[k];
          nff1[k] = (nf1[nv1[k]]+nf1[nv3[k]])/2.000 -
            cfl * (nf1[nv1[k]]-nf1[nv3[k]])/2.000 + source1[k];
          nff2[k] = nf2[k] + cfl * (nf2[nv4[k]]-nf2[k]) + source2[k];
          nff3[k] = (nf3[nv3[k]]+nf3[nv1[k]])/2.000 -
            cfl * (nf3[nv3[k]]-nf3[nv1[k]])/2.000 + source3[k];
          nff4[k] = nf4[k] - cfl * (nf4[nv4[k]]-nf4[k]) + source4[k];
          nff5[k] = nf5[k] + cfl * (nf5[nv7[k]]-nf5[k]) + source5[k];
          nff6[k] = nf6[k] + cfl * (nf6[nv8[k]]-nf6[k]) + source6[k];
          nff7[k] = nf7[k] - cfl * (nf7[nv7[k]]-nf7[k]) + source7[k];
          nff8[k] = nf8[k] - cfl * (nf8[nv8[k]]-nf8[k]) + source8[k];
          break;
        }
    }
}
```

```c
/**********************************************************************\
*                                                                      *
*   wet9lf.c                                                           *
*                                                                      *
\**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "wet9head.h"

double compute_upwind_sources_bulk(int sigma, int index, int k, int nv,
                                   double nf10[], double nf11[], double nf12[],
                                   double nf13[], double nf14[], double nf15[],
                                   double nf16[], double nf17[], double nf18[],
                                   double nf20[], double nf21[], double nf22[],
                                   double nf23[], double nf24[], double nf25[],
                                   double nf26[], double nf27[], double nf28[]);

double compute_upwind_sources_boundary(int sigma, int index, int k, int nv,
                                   double ux_boundary, double uy_boundary,
                                   double nf10[], double nf11[], double nf12[],
                                   double nf13[], double nf14[], double nf15[],
                                   double nf16[], double nf17[], double nf18[],
                                   double nf20[], double nf21[], double nf22[],
                                   double nf23[], double nf24[], double nf25[],
                                   double nf26[], double nf27[], double nf28[]);

double lf_sources_bulk(int sigma, int index,
                       int jminus, int jloc, int jplus,
                       double tlfminus, double tlfplus,
                       double nf10[], double nf11[], double nf12[],
                       double nf13[], double nf14[], double nf15[],
                       double nf16[], double nf17[], double nf18[],
                       double nf20[], double nf21[], double nf22[],
                       double nf23[], double nf24[], double nf25[],
                       double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, source;
  double fs1[9], fs2[9];

  fs1[0] = tlfminus * nf10[jminus] + tlfplus * nf10[jplus];
  fs1[1] = tlfminus * nf11[jminus] + tlfplus * nf11[jplus];
  fs1[2] = tlfminus * nf12[jminus] + tlfplus * nf12[jplus];
  fs1[3] = tlfminus * nf13[jminus] + tlfplus * nf13[jplus];
  fs1[4] = tlfminus * nf14[jminus] + tlfplus * nf14[jplus];
  fs1[5] = tlfminus * nf15[jminus] + tlfplus * nf15[jplus];
  fs1[6] = tlfminus * nf16[jminus] + tlfplus * nf16[jplus];
  fs1[7] = tlfminus * nf17[jminus] + tlfplus * nf17[jplus];
  fs1[8] = tlfminus * nf18[jminus] + tlfplus * nf18[jplus];

  fs2[0] = tlfminus * nf20[jminus] + tlfplus * nf20[jplus];
  fs2[1] = tlfminus * nf21[jminus] + tlfplus * nf21[jplus];
  fs2[2] = tlfminus * nf22[jminus] + tlfplus * nf22[jplus];
  fs2[3] = tlfminus * nf23[jminus] + tlfplus * nf23[jplus];
  fs2[4] = tlfminus * nf24[jminus] + tlfplus * nf24[jplus];
  fs2[5] = tlfminus * nf25[jminus] + tlfplus * nf25[jplus];
  fs2[6] = tlfminus * nf26[jminus] + tlfplus * nf26[jplus];
  fs2[7] = tlfminus * nf27[jminus] + tlfplus * nf27[jplus];
  fs2[8] = tlfminus * nf28[jminus] + tlfplus * nf28[jplus];

  dummy = mass1 * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+
                   fs1[6]+fs1[7]+fs1[8]) / tau1 +
          mass2 * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+
                   fs2[6]+fs2[7]+fs2[8]) / tau2;
  ux = (mass1 * (fs1[1]*ecx1[1]+fs1[2]*ecx1[2]+fs1[3]*ecx1[3]+fs1[4]*ecx1[4]+
                 fs1[5]*ecx1[5]+fs1[6]*ecx1[6]+fs1[7]*ecx1[7]+fs1[8]*ecx1[8])
        /tau1 +
```



```c
        mass2 * (fs2[1]*ecx2[1]+fs2[2]*ecx2[2]+fs2[3]*ecx2[3]+fs2[4]*ecx2[4]+
                 fs2[5]*ecx2[5]+fs2[6]*ecx2[6]+fs2[7]*ecx2[7]+fs2[8]*ecx2[8])
        /tau2) / dummy;
  uy = (mass1 * (fs1[1]*ecy1[1]+fs1[2]*ecy1[2]+fs1[3]*ecy1[3]+fs1[4]*ecy1[4]+
                 fs1[5]*ecy1[5]+fs1[6]*ecy1[6]+fs1[7]*ecy1[7]+fs1[8]*ecy1[8])
        /tau1 +
        mass2 * (fs2[1]*ecy2[1]+fs2[2]*ecy2[2]+fs2[3]*ecy2[3]+fs2[4]*ecy2[4]+
                 fs2[5]*ecy2[5]+fs2[6]*ecy2[6]+fs2[7]*ecy2[7]+fs2[8]*ecy2[8])
        /tau2) / dummy;

  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    }
  return source;
}

double lf_sources_out(int sigma, int index,
                      int jminus2, int jminus1, int jloc,
                      double tminus2, double tminus1, double tout,
                      double nf10[], double nf11[], double nf12[],
                      double nf13[], double nf14[], double nf15[],
                      double nf16[], double nf17[], double nf18[],
                      double nf20[], double nf21[], double nf22[],
                      double nf23[], double nf24[], double nf25[],
                      double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, source;
  double fs1[9], fs2[9];

  fs1[0] = tminus2 * nf10[jminus2] + tminus1 * nf10[jminus1] + tout*nf10[jloc];
  fs1[1] = tminus2 * nf11[jminus2] + tminus1 * nf11[jminus1] + tout*nf11[jloc];
  fs1[2] = tminus2 * nf12[jminus2] + tminus1 * nf12[jminus1] + tout*nf12[jloc];
  fs1[3] = tminus2 * nf13[jminus2] + tminus1 * nf13[jminus1] + tout*nf13[jloc];
  fs1[4] = tminus2 * nf14[jminus2] + tminus1 * nf14[jminus1] + tout*nf14[jloc];
  fs1[5] = tminus2 * nf15[jminus2] + tminus1 * nf15[jminus1] + tout*nf15[jloc];
  fs1[6] = tminus2 * nf16[jminus2] + tminus1 * nf16[jminus1] + tout*nf16[jloc];
  fs1[7] = tminus2 * nf17[jminus2] + tminus1 * nf17[jminus1] + tout*nf17[jloc];
  fs1[8] = tminus2 * nf18[jminus2] + tminus1 * nf18[jminus1] + tout*nf18[jloc];

  fs2[0] = tminus2 * nf20[jminus2] + tminus1 * nf20[jminus1] + tout*nf20[jloc];
  fs2[1] = tminus2 * nf21[jminus2] + tminus1 * nf21[jminus1] + tout*nf21[jloc];
  fs2[2] = tminus2 * nf22[jminus2] + tminus1 * nf22[jminus1] + tout*nf22[jloc];
  fs2[3] = tminus2 * nf23[jminus2] + tminus1 * nf23[jminus1] + tout*nf23[jloc];
  fs2[4] = tminus2 * nf24[jminus2] + tminus1 * nf24[jminus1] + tout*nf24[jloc];
  fs2[5] = tminus2 * nf25[jminus2] + tminus1 * nf25[jminus1] + tout*nf25[jloc];
  fs2[6] = tminus2 * nf26[jminus2] + tminus1 * nf26[jminus1] + tout*nf26[jloc];
  fs2[7] = tminus2 * nf27[jminus2] + tminus1 * nf27[jminus1] + tout*nf27[jloc];
  fs2[8] = tminus2 * nf28[jminus2] + tminus1 * nf28[jminus1] + tout*nf28[jloc];

  ux = tminus2 * uxloc[jminus2] + tminus1 * uxloc[jminus1] + tout *uxloc[jloc];
```

```c
  uy = tminus2 * uyloc[jminus2] + tminus1 * uyloc[jminus1] + tout *uyloc[jloc];

  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    }
  return source;
}

double lf_sources_in(int sigma, int index,
                     int jloc, int jplus1, int jplus2,
                     double tin, double tplus1, double tplus2,
                     double nf10[], double nf11[], double nf12[],
                     double nf13[], double nf14[], double nf15[],
                     double nf16[], double nf17[], double nf18[],
                     double nf20[], double nf21[], double nf22[],
                     double nf23[], double nf24[], double nf25[],
                     double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, source;
  double fs1[9], fs2[9];

  fs1[0] = tin * nf10[jloc] + tplus1 * nf10[jplus1] + tplus2 * nf10[jplus2];
  fs1[1] = tin * nf11[jloc] + tplus1 * nf11[jplus1] + tplus2 * nf11[jplus2];
  fs1[2] = tin * nf12[jloc] + tplus1 * nf12[jplus1] + tplus2 * nf12[jplus2];
  fs1[3] = tin * nf13[jloc] + tplus1 * nf13[jplus1] + tplus2 * nf13[jplus2];
  fs1[4] = tin * nf14[jloc] + tplus1 * nf14[jplus1] + tplus2 * nf14[jplus2];
  fs1[5] = tin * nf15[jloc] + tplus1 * nf15[jplus1] + tplus2 * nf15[jplus2];
  fs1[6] = tin * nf16[jloc] + tplus1 * nf16[jplus1] + tplus2 * nf16[jplus2];
  fs1[7] = tin * nf17[jloc] + tplus1 * nf17[jplus1] + tplus2 * nf17[jplus2];
  fs1[8] = tin * nf18[jloc] + tplus1 * nf18[jplus1] + tplus2 * nf18[jplus2];

  fs2[0] = tin * nf20[jloc] + tplus1 * nf20[jplus1] + tplus2 * nf20[jplus2];
  fs2[1] = tin * nf21[jloc] + tplus1 * nf21[jplus1] + tplus2 * nf21[jplus2];
  fs2[2] = tin * nf22[jloc] + tplus1 * nf22[jplus1] + tplus2 * nf22[jplus2];
  fs2[3] = tin * nf23[jloc] + tplus1 * nf23[jplus1] + tplus2 * nf23[jplus2];
  fs2[4] = tin * nf24[jloc] + tplus1 * nf24[jplus1] + tplus2 * nf24[jplus2];
  fs2[5] = tin * nf25[jloc] + tplus1 * nf25[jplus1] + tplus2 * nf25[jplus2];
  fs2[6] = tin * nf26[jloc] + tplus1 * nf26[jplus1] + tplus2 * nf26[jplus2];
  fs2[7] = tin * nf27[jloc] + tplus1 * nf27[jplus1] + tplus2 * nf27[jplus2];
  fs2[8] = tin * nf28[jloc] + tplus1 * nf28[jplus1] + tplus2 * nf28[jplus2];

  ux = tin * uxloc[jloc] + tplus1 * uxloc[jplus1] + tplus2 * uxloc[jplus2];
  uy = tin * uyloc[jloc] + tplus1 * uyloc[jplus1] + tplus2 * uyloc[jplus2];

  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
```



```c
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    }
  return source;
}

void lf_sources(double nf10[], double nf11[], double nf12[],
                double nf13[], double nf14[], double nf15[],
                double nf16[], double nf17[], double nf18[],
                double nf20[], double nf21[], double nf22[],
                double nf23[], double nf24[], double nf25[],
                double nf26[], double nf27[], double nf28[])
{
  int k;

  for(k=0; k<nnodes_all; k++)
    {
      sf10[k] = nf10[k];
      sf20[k] = nf20[k];
      source10[k] = - ctau1 * (nf10[k] - neq10[k]) -
        ( csforce_x * uxloc[k] + csforce_y * uyloc[k] ) * neq10[k];
      source20[k] = - ctau2 * (nf20[k] - neq20[k]) -
        ( csforce_x * uxloc[k] + csforce_y * uyloc[k] ) * neq20[k];

      switch(boundary_mode[k])
        {
        case 0:
          sf11[k] =  tlfminus1 * nf11[nv3[k]] + tlfplus1 * nf11[nv1[k]];
          source11[k] = lf_sources_bulk(1, 1, nv3[k], k, nv1[k],
                        tlfminus1, tlfplus1,
                        nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                        nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
          sf21[k] =  tlfminus2 * nf21[nv3[k]] + tlfplus2 * nf21[nv1[k]];
          source21[k] = lf_sources_bulk(2, 1, nv3[k], k, nv1[k],
                        tlfminus2, tlfplus2,
                        nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                        nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

          sf12[k] =  tlfminus1 * nf12[nv4[k]] + tlfplus1 * nf12[nv2[k]];
          source12[k] = lf_sources_bulk(1, 2, nv4[k], k, nv2[k],
                        tlfminus1, tlfplus1,
                        nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                        nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
          sf22[k] =  tlfminus2 * nf22[nv4[k]] + tlfplus2 * nf22[nv2[k]];
          source22[k] = lf_sources_bulk(2, 2, nv4[k], k, nv2[k],
                        tlfminus2, tlfplus2,
                        nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                        nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

          sf13[k] =  tlfminus1 * nf13[nv1[k]] + tlfplus1 * nf13[nv3[k]];
          source13[k] = lf_sources_bulk(1, 3, nv1[k], k, nv3[k],
                        tlfminus1, tlfplus1,
                        nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                        nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
          sf23[k] =  tlfminus2 * nf23[nv1[k]] + tlfplus2 * nf23[nv3[k]];
          source23[k] = lf_sources_bulk(2, 3, nv1[k], k, nv3[k],
```

```
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf14[k] =  tlfminus1 * nf14[nv2[k]] + tlfplus1 * nf14[nv4[k]];
    source14[k] = lf_sources_bulk(1, 4, nv2[k], k, nv4[k],
                     tlfminus1, tlfplus1,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf24[k] =  tlfminus2 * nf24[nv2[k]] + tlfplus2 * nf24[nv4[k]];
    source24[k] = lf_sources_bulk(2, 4, nv2[k], k, nv4[k],
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf15[k] =  tlfminus1 * nf15[nv7[k]] + tlfplus1 * nf15[nv5[k]];
    source15[k] = lf_sources_bulk(1, 5, nv7[k], k, nv5[k],
                     tlfminus1, tlfplus1,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf25[k] =  tlfminus2 * nf25[nv7[k]] + tlfplus2 * nf25[nv5[k]];
    source25[k] = lf_sources_bulk(2, 5, nv7[k], k, nv5[k],
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf16[k] =  tlfminus1 * nf16[nv8[k]] + tlfplus1 * nf16[nv6[k]];
    source16[k] = lf_sources_bulk(1, 6, nv8[k], k, nv6[k],
                     tlfminus1, tlfplus1,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf26[k] =  tlfminus2 * nf26[nv8[k]] + tlfplus2 * nf26[nv6[k]];
    source26[k] = lf_sources_bulk(2, 6, nv8[k], k, nv6[k],
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf17[k] =  tlfminus1 * nf17[nv5[k]] + tlfplus1 * nf17[nv7[k]];
    source17[k] = lf_sources_bulk(1, 7, nv5[k], k, nv7[k],
                     tlfminus1, tlfplus1,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf27[k] =  tlfminus2 * nf27[nv5[k]] + tlfplus2 * nf27[nv7[k]];
    source27[k] = lf_sources_bulk(2, 7, nv5[k], k, nv7[k],
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf18[k] =  tlfminus1 * nf18[nv6[k]] + tlfplus1 * nf18[nv8[k]];
    source18[k] = lf_sources_bulk(1, 8, nv6[k], k, nv8[k],
                     tlfminus1, tlfplus1,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf28[k] =  tlfminus2 * nf28[nv6[k]] + tlfplus2 * nf28[nv8[k]];
    source28[k] = lf_sources_bulk(2, 8, nv6[k], k, nv8[k],
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    break;
  case 1:  /*  bottom wall  */
    sf11[k] =  tlfminus1 * nf11[nv3[k]] + tlfplus1 * nf11[nv1[k]];
    source11[k] = lf_sources_bulk(1, 1, nv3[k], k, nv1[k],
                     tlfminus1, tlfplus1,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf21[k] =  tlfminus2 * nf21[nv3[k]] + tlfplus2 * nf21[nv1[k]];
    source21[k] = lf_sources_bulk(2, 1, nv3[k], k, nv1[k],
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
```



```
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf12[k] = nf12[k] - cf11 * (nf12[nv2[k]] - nf12[k]);
    source12[k] = compute_upwind_sources_boundary(1, 2, k, nv2[k],
                     uxwall_bot, uywall_bot,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf22[k] = nf22[k] - cf12 * (nf22[nv2[k]] - nf22[k]);
    source22[k] = compute_upwind_sources_boundary(2, 2, k, nv2[k],
                     uxwall_bot, uywall_bot,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf13[k] =  tlfminus1 * nf13[nv1[k]] + tlfplus1 * nf13[nv3[k]];
    source13[k] = lf_sources_bulk(1, 3, nv1[k], k, nv3[k],
                     tlfminus1, tlfplus1,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf23[k] =  tlfminus1 * nf23[nv1[k]] + tlfplus2 * nf23[nv3[k]];
    source23[k] = lf_sources_bulk(2, 3, nv1[k], k, nv3[k],
                     tlfminus2, tlfplus2,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]);
    source14[k] = compute_upwind_sources_bulk(1, 4, k, nv2[k],
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf24[k] = nf24[k] - cf12 * (nf24[k] - nf24[nv2[k]]);
    source24[k] = compute_upwind_sources_bulk(2, 4, k, nv2[k],
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf15[k] = nf15[k] - cf11 * (nf15[nv5[k]] - nf15[k]);
    source15[k] = compute_upwind_sources_boundary(1, 5, k, nv5[k],
                     uxwall_bot, uywall_bot,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf25[k] = nf25[k] - cf12 * (nf25[nv5[k]] - nf25[k]);
    source25[k] = compute_upwind_sources_boundary(2, 5, k, nv5[k],
                     uxwall_bot, uywall_bot,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf16[k] = nf16[k] - cf11 * (nf16[nv6[k]] - nf16[k]);
    source16[k] = compute_upwind_sources_boundary(1, 6, k, nv6[k],
                     uxwall_bot, uywall_bot,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf26[k] = nf26[k] - cf12 * (nf26[nv6[k]] - nf26[k]);
    source26[k] = compute_upwind_sources_boundary(2, 6, k, nv6[k],
                     uxwall_bot, uywall_bot,
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]);
    source17[k] = compute_upwind_sources_bulk(1, 7, k, nv5[k],
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]);
    source27[k] = compute_upwind_sources_bulk(2, 7, k, nv5[k],
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]);
    source18[k] = compute_upwind_sources_bulk(1, 8, k, nv6[k],
                     nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                     nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf28[k] = nf28[k] - cf12 * (nf28[k] - nf28[nv6[k]]);
```

```
      source28[k] = compute_upwind_sources_bulk(2, 8, k, nv6[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      break;
    case 2:  /*  top wall  */
      sf11[k] =  tlfminus1 * nf11[nv3[k]] + tlfplus1 * nf11[nv1[k]];
      source11[k] = lf_sources_bulk(1, 1, nv3[k], k, nv1[k],
                    tlfminus1, tlfplus1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf21[k] =  tlfminus2 * nf21[nv3[k]] + tlfplus2 * nf21[nv1[k]];
      source21[k] = lf_sources_bulk(2, 1, nv3[k], k, nv1[k],
                    tlfminus2, tlfplus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf12[k] = nf12[k] - cfl1 * (nf12[k] - nf12[nv4[k]]);
      source12[k] = compute_upwind_sources_bulk(1, 2, k, nv4[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf22[k] = nf22[k] - cfl2 * (nf22[k] - nf22[nv4[k]]);
      source22[k] = compute_upwind_sources_bulk(2, 2, k, nv4[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf13[k] =  tlfminus1 * nf13[nv1[k]] + tlfplus1 * nf13[nv3[k]];
      source13[k] = lf_sources_bulk(1, 3, nv1[k], k, nv3[k],
                    tlfminus1, tlfplus1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf23[k] =  tlfminus2 * nf23[nv1[k]] + tlfplus2 * nf23[nv3[k]];
      source23[k] = lf_sources_bulk(2, 3, nv1[k], k, nv3[k],
                    tlfminus2, tlfplus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf14[k] = nf14[k] - cfl1 * (nf14[nv4[k]] - nf14[k]);
      source14[k] = compute_upwind_sources_boundary(1, 4, k, nv4[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf24[k] = nf24[k] - cfl2 * (nf24[nv4[k]] - nf24[k]);
      source24[k] = compute_upwind_sources_boundary(2, 4, k, nv4[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf15[k] = nf15[k] - cfl1 * (nf15[k] - nf15[nv7[k]]);
      source15[k] = compute_upwind_sources_bulk(1, 5, k, nv7[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf25[k] = nf25[k] - cfl2 * (nf25[k] - nf25[nv7[k]]);
      source25[k] = compute_upwind_sources_bulk(2, 5, k, nv7[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf16[k] = nf16[k] - cfl1 * (nf16[k] - nf16[nv8[k]]);
      source16[k] = compute_upwind_sources_bulk(1, 6, k, nv8[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf26[k] = nf26[k] - cfl2 * (nf26[k] - nf26[nv8[k]]);
      source26[k] = compute_upwind_sources_bulk(2, 6, k, nv8[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf17[k] = nf17[k] - cfl1 * (nf17[nv7[k]] - nf17[k]);
      source17[k] = compute_upwind_sources_boundary(1, 7, k, nv7[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
```



```
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf27[k] = nf27[k] - cfl2 * (nf27[nv7[k]] - nf27[k]);
      source27[k] = compute_upwind_sources_boundary(2, 7, k, nv7[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf18[k] = nf18[k] - cfl1 * (nf18[nv8[k]] - nf18[k]);
      source18[k] = compute_upwind_sources_boundary(1, 8, k, nv8[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf28[k] = nf28[k] - cfl2 * (nf28[nv8[k]] - nf28[k]);
      source28[k] = compute_upwind_sources_boundary(2, 8, k, nv8[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      break;
    case 3:
      sf11[k] = nf11[k] - cfl1 * (nf11[nv1[k]] - nf11[k]);
      source11[k] = compute_upwind_sources_boundary(1, 1, k, nv1[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf21[k] = nf21[k] - cfl2 * (nf21[nv1[k]] - nf21[k]);
      source21[k] = compute_upwind_sources_boundary(2, 1, k, nv1[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf12[k] =  tlfminus1 * nf12[nv4[k]] + tlfplus1 * nf12[nv2[k]];
      source12[k] = lf_sources_bulk(1, 2, nv4[k], k, nv2[k],
                    tlfminus1, tlfplus1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf22[k] =  tlfminus2 * nf22[nv4[k]] + tlfplus2 * nf22[nv2[k]];
      source22[k] = lf_sources_bulk(2, 2, nv4[k], k, nv2[k],
                    tlfminus2, tlfplus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf13[k] = nf13[k] - cfl1 * (nf13[k] - nf13[nv1[k]]);
      source13[k] = compute_upwind_sources_bulk(1, 3, k, nv1[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf23[k] = nf23[k] - cfl2 * (nf23[k] - nf23[nv1[k]]);
      source23[k] = compute_upwind_sources_bulk(2, 3, k, nv1[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf14[k] =  tlfminus1 * nf14[nv2[k]] + tlfplus1 * nf14[nv4[k]];
      source14[k] = lf_sources_bulk(1, 4, nv2[k], k, nv4[k],
                    tlfminus1, tlfplus1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf24[k] =  tlfminus2 * nf24[nv2[k]] + tlfplus2 * nf24[nv4[k]];
      source24[k] = lf_sources_bulk(2, 4, nv2[k], k, nv4[k],
                    tlfminus2, tlfplus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf15[k] = nf15[k] - cfl1 * (nf15[nv5[k]] - nf15[k]);
      source15[k] = compute_upwind_sources_boundary(1, 5, k, nv5[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf25[k] = nf25[k] - cfl2 * (nf25[nv5[k]] - nf25[k]);
      source25[k] = compute_upwind_sources_boundary(2, 5, k, nv5[k],
                    uxwall_left, uywall_left,
```

```
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf16[k] = nf16[k] - cf11 * (nf16[k] - nf16[nv8[k]]);
    source16[k] = compute_upwind_sources_bulk(1, 6, k, nv8[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf26[k] = nf26[k] - cf12 * (nf26[k] - nf26[nv8[k]]);
    source26[k] = compute_upwind_sources_bulk(2, 6, k, nv8[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]);
    source17[k] = compute_upwind_sources_bulk(1, 7, k, nv5[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]);
    source27[k] = compute_upwind_sources_bulk(2, 7, k, nv5[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf18[k] = nf18[k] - cf11 * (nf18[nv8[k]] - nf18[k]);
    source18[k] = compute_upwind_sources_boundary(1, 8, k, nv8[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf28[k] = nf28[k] - cf12 * (nf28[nv8[k]] - nf28[k]);
    source28[k] = compute_upwind_sources_boundary(2, 8, k, nv8[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    break;
  case 4:
    sf11[k] = nf11[k] - cf11 * (nf11[k] - nf11[nv3[k]]);
    source13[k] = compute_upwind_sources_bulk(1, 1, k, nv3[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf21[k] = nf21[k] - cf12 * (nf21[k] - nf21[nv3[k]]);
    source21[k] = compute_upwind_sources_bulk(2, 1, k, nv3[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf12[k] =  tlfminus1 * nf12[nv4[k]] + tlfplus1 * nf12[nv2[k]];
    source12[k] = lf_sources_bulk(1, 2, nv4[k], k, nv2[k],
                    tlfminus1, tlfplus1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf22[k] =  tlfminus2 * nf22[nv4[k]] +  tlfplus2 * nf22[nv2[k]];
    source22[k] = lf_sources_bulk(2, 2, nv4[k], k, nv2[k],
                    tlfminus2, tlfplus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf13[k] = nf13[k] - cf11 * (nf13[nv3[k]] - nf13[k]);
    source13[k] = compute_upwind_sources_boundary(1, 3, k, nv3[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf23[k] = nf23[k] - cf12 * (nf23[nv3[k]] - nf23[k]);
    source23[k] = compute_upwind_sources_boundary(2, 3, k, nv3[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf14[k] =  tlfminus1 * nf14[nv2[k]] + tlfplus1 * nf14[nv4[k]];
    source14[k] = lf_sources_bulk(1, 4, nv2[k], k, nv4[k],
                    tlfminus1, tlfplus1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
```



```
    sf24[k] =  tlfminus2 * nf24[nv2[k]] + tlfplus2 * nf24[nv4[k]];
    source24[k] = lf_sources_bulk(2, 4, nv2[k], k, nv4[k],
                    tlfminus2, tlfplus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]);
    source15[k] = compute_upwind_sources_bulk(1, 5, k, nv7[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf25[k] = nf25[k] - cf12 * (nf25[k] - nf25[nv7[k]]);
    source25[k] = compute_upwind_sources_bulk(2, 5, k, nv7[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf16[k] = nf16[k] - cf11 * (nf16[nv6[k]] - nf16[k]);
    source16[k] = compute_upwind_sources_boundary(1, 6, k, nv6[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf26[k] = nf26[k] - cf12 * (nf26[nv6[k]] - nf26[k]);
    source26[k] = compute_upwind_sources_boundary(2, 6, k, nv6[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf17[k] = nf17[k] - cf11 * (nf17[nv7[k]] - nf17[k]);
    source17[k] = compute_upwind_sources_boundary(1, 7, k, nv7[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf27[k] = nf27[k] - cf12 * (nf27[nv7[k]] - nf27[k]);
    source27[k] = compute_upwind_sources_boundary(2, 7, k, nv7[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]);
    source18[k] = compute_upwind_sources_bulk(1, 8, k, nv6[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf28[k] = nf28[k] - cf12 * (nf28[k] - nf28[nv6[k]]);
    source28[k] = compute_upwind_sources_bulk(2, 8, k, nv6[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    break;
   }
  }
}

void lf(double sf0[], double sf1[], double sf2[],
        double sf3[], double sf4[], double sf5[],
        double sf6[], double sf7[], double sf8[],
        double source0[], double source1[], double source2[],
        double source3[], double source4[], double source5[],
        double source6[], double source7[], double source8[],
        double nff0[], double nff1[], double nff2[],
        double nff3[], double nff4[], double nff5[],
        double nff6[], double nff7[], double nff8[])
{
  int k;

  for(k=0; k<nnodes_all; k++)
    {
      nff0[k] = sf0[k] + source0[k];
      nff1[k] = sf1[k] + source1[k];
      nff2[k] = sf2[k] + source2[k];
      nff3[k] = sf3[k] + source3[k];
      nff4[k] = sf4[k] + source4[k];
```

```
        nff5[k] = sf5[k] + source5[k];
        nff6[k] = sf6[k] + source6[k];
        nff7[k] = sf7[k] + source7[k];
        nff8[k] = sf8[k] + source8[k];
      }
  }
```

```c
/**********************************************************************\
*                                                                      *
*   wet9lw.c                                                           *
*                                                                      *
\**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "wet9head.h"

double compute_upwind_sources_bulk(int sigma, int index, int k, int nv,
                                   double nf10[], double nf11[], double nf12[],
                                   double nf13[], double nf14[], double nf15[],
                                   double nf16[], double nf17[], double nf18[],
                                   double nf20[], double nf21[], double nf22[],
                                   double nf23[], double nf24[], double nf25[],
                                   double nf26[], double nf27[], double nf28[]);

double compute_upwind_sources_boundary(int sigma, int index, int k, int nv,
                                   double ux_boundary, double uy_boundary,
                                   double nf10[], double nf11[], double nf12[],
                                   double nf13[], double nf14[], double nf15[],
                                   double nf16[], double nf17[], double nf18[],
                                   double nf20[], double nf21[], double nf22[],
                                   double nf23[], double nf24[], double nf25[],
                                   double nf26[], double nf27[], double nf28[]);

double compute_upwind_sources_boundary_in(int sigma, int index, int k, int nv,
                                   double ux_boundary, double uy_boundary,
                                   double nf10[], double nf11[], double nf12[],
                                   double nf13[], double nf14[], double nf15[],
                                   double nf16[], double nf17[], double nf18[],
                                   double nf20[], double nf21[], double nf22[],
                                   double nf23[], double nf24[], double nf25[],
                                   double nf26[], double nf27[], double nf28[]);

double compute_upwind_sources_boundary_out(int sigma, int index, int nv, int k,
                                   double ux_boundary, double uy_boundary,
                                   double nf10[], double nf11[], double nf12[],
                                   double nf13[], double nf14[], double nf15[],
                                   double nf16[], double nf17[], double nf18[],
                                   double nf20[], double nf21[], double nf22[],
                                   double nf23[], double nf24[], double nf25[],
                                   double nf26[], double nf27[], double nf28[]);

double lw_sources_bulk(int sigma, int index,
                       int jminus, int jloc, int jplus,
                       double tminus, double tloc, double tplus,
                       double nf10[], double nf11[], double nf12[],
                       double nf13[], double nf14[], double nf15[],
                       double nf16[], double nf17[], double nf18[],
                       double nf20[], double nf21[], double nf22[],
                       double nf23[], double nf24[], double nf25[],
                       double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, source;
  double fs1[9], fs2[9];

  fs1[0] = tminus * nf10[jminus] + tloc * nf10[jloc] + tplus * nf10[jplus];
  fs1[1] = tminus * nf11[jminus] + tloc * nf11[jloc] + tplus * nf11[jplus];
  fs1[2] = tminus * nf12[jminus] + tloc * nf12[jloc] + tplus * nf12[jplus];
  fs1[3] = tminus * nf13[jminus] + tloc * nf13[jloc] + tplus * nf13[jplus];
  fs1[4] = tminus * nf14[jminus] + tloc * nf14[jloc] + tplus * nf14[jplus];
  fs1[5] = tminus * nf15[jminus] + tloc * nf15[jloc] + tplus * nf15[jplus];
  fs1[6] = tminus * nf16[jminus] + tloc * nf16[jloc] + tplus * nf16[jplus];
  fs1[7] = tminus * nf17[jminus] + tloc * nf17[jloc] + tplus * nf17[jplus];
  fs1[8] = tminus * nf18[jminus] + tloc * nf18[jloc] + tplus * nf18[jplus];
```



```c
  fs2[0] = tminus * nf20[jminus] + tloc * nf20[jloc] + tplus * nf20[jplus];
  fs2[1] = tminus * nf21[jminus] + tloc * nf21[jloc] + tplus * nf21[jplus];
  fs2[2] = tminus * nf22[jminus] + tloc * nf22[jloc] + tplus * nf22[jplus];
  fs2[3] = tminus * nf23[jminus] + tloc * nf23[jloc] + tplus * nf23[jplus];
  fs2[4] = tminus * nf24[jminus] + tloc * nf24[jloc] + tplus * nf24[jplus];
  fs2[5] = tminus * nf25[jminus] + tloc * nf25[jloc] + tplus * nf25[jplus];
  fs2[6] = tminus * nf26[jminus] + tloc * nf26[jloc] + tplus * nf26[jplus];
  fs2[7] = tminus * nf27[jminus] + tloc * nf27[jloc] + tplus * nf27[jplus];
  fs2[8] = tminus * nf28[jminus] + tloc * nf28[jloc] + tplus * nf28[jplus];

  switch(boundary_mode[jloc])
   {
    case 0:
      dummy = mass1 * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+
                       fs1[6]+fs1[7]+fs1[8]) / tau1 +
        mass2 * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+
                 fs2[6]+fs2[7]+fs2[8]) / tau2;
      ux = (mass1 * (fs1[1]*ecx1[1]+fs1[2]*ecx1[2]+fs1[3]*ecx1[3]+
                     fs1[4]*ecx1[4]+fs1[5]*ecx1[5]+fs1[6]*ecx1[6]+
                     fs1[7]*ecx1[7]+fs1[8]*ecx1[8])/tau1 +
            mass2 * (fs2[1]*ecx2[1]+fs2[2]*ecx2[2]+fs2[3]*ecx2[3]+
                     fs2[4]*ecx2[4]+fs2[5]*ecx2[5]+fs2[6]*ecx2[6]+
                     fs2[7]*ecx2[7]+fs2[8]*ecx2[8])/tau2) / dummy;
      uy = (mass1 * (fs1[1]*ecy1[1]+fs1[2]*ecy1[2]+fs1[3]*ecy1[3]+
                     fs1[4]*ecy1[4]+fs1[5]*ecy1[5]+fs1[6]*ecy1[6]+
                     fs1[7]*ecy1[7]+fs1[8]*ecy1[8])/tau1 +
            mass2 * (fs2[1]*ecy2[1]+fs2[2]*ecy2[2]+fs2[3]*ecy2[3]+
                     fs2[4]*ecy2[4]+fs2[5]*ecy2[5]+fs2[6]*ecy2[6]+
                     fs2[7]*ecy2[7]+fs2[8]*ecy2[8])/tau2) / dummy;
  break;
    case 1:
      ux = uxwall_bot;
      uy = uywall_bot;
      break;
    case 2:
      ux = uxwall_top;
      uy = uywall_top;
      break;
   }

  uu = ux*ux + uy*uy;
  switch(sigma)
   {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
   }
  return source;
}


double lw_sources_out(int sigma, int index,
                      int jminus2, int jminus1, int jloc,
```

```
                    double tminus2, double tminus1, double tout,
                    double nf10[], double nf11[], double nf12[],
                    double nf13[], double nf14[], double nf15[],
                    double nf16[], double nf17[], double nf18[],
                    double nf20[], double nf21[], double nf22[],
                    double nf23[], double nf24[], double nf25[],
                    double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, source;
  double fs1[9], fs2[9];

  fs1[0] = tminus2 * nf10[jminus2] + tminus1 * nf10[jminus1] + tout*nf10[jloc];
  fs1[1] = tminus2 * nf11[jminus2] + tminus1 * nf11[jminus1] + tout*nf11[jloc];
  fs1[2] = tminus2 * nf12[jminus2] + tminus1 * nf12[jminus1] + tout*nf12[jloc];
  fs1[3] = tminus2 * nf13[jminus2] + tminus1 * nf13[jminus1] + tout*nf13[jloc];
  fs1[4] = tminus2 * nf14[jminus2] + tminus1 * nf14[jminus1] + tout*nf14[jloc];
  fs1[5] = tminus2 * nf15[jminus2] + tminus1 * nf15[jminus1] + tout*nf15[jloc];
  fs1[6] = tminus2 * nf16[jminus2] + tminus1 * nf16[jminus1] + tout*nf16[jloc];
  fs1[7] = tminus2 * nf17[jminus2] + tminus1 * nf17[jminus1] + tout*nf17[jloc];
  fs1[8] = tminus2 * nf18[jminus2] + tminus1 * nf18[jminus1] + tout*nf18[jloc];

  fs2[0] = tminus2 * nf20[jminus2] + tminus1 * nf20[jminus1] + tout*nf20[jloc];
  fs2[1] = tminus2 * nf21[jminus2] + tminus1 * nf21[jminus1] + tout*nf21[jloc];
  fs2[2] = tminus2 * nf22[jminus2] + tminus1 * nf22[jminus1] + tout*nf22[jloc];
  fs2[3] = tminus2 * nf23[jminus2] + tminus1 * nf23[jminus1] + tout*nf23[jloc];
  fs2[4] = tminus2 * nf24[jminus2] + tminus1 * nf24[jminus1] + tout*nf24[jloc];
  fs2[5] = tminus2 * nf25[jminus2] + tminus1 * nf25[jminus1] + tout*nf25[jloc];
  fs2[6] = tminus2 * nf26[jminus2] + tminus1 * nf26[jminus1] + tout*nf26[jloc];
  fs2[7] = tminus2 * nf27[jminus2] + tminus1 * nf27[jminus1] + tout*nf27[jloc];
  fs2[8] = tminus2 * nf28[jminus2] + tminus1 * nf28[jminus1] + tout*nf28[jloc];

  ux = tminus2 * uxloc[jminus2] + tminus1 * uxloc[jminus1] + tout *uxloc[jloc];
  uy = tminus2 * uyloc[jminus2] + tminus1 * uyloc[jminus1] + tout *uyloc[jloc];
  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    }
  return source;
}

double lw_sources_in(int sigma, int index,
                    int jloc, int jplus1, int jplus2,
                    double tin, double tplus1, double tplus2,
                    double nf10[], double nf11[], double nf12[],
                    double nf13[], double nf14[], double nf15[],
                    double nf16[], double nf17[], double nf18[],
                    double nf20[], double nf21[], double nf22[],
                    double nf23[], double nf24[], double nf25[],
                    double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, source;
```



```
  double fs1[9], fs2[9];

  fs1[0] = tin * nf10[jloc] + tplus1 * nf10[jplus1] + tplus2 * nf10[jplus2];
  fs1[1] = tin * nf11[jloc] + tplus1 * nf11[jplus1] + tplus2 * nf11[jplus2];
  fs1[2] = tin * nf12[jloc] + tplus1 * nf12[jplus1] + tplus2 * nf12[jplus2];
  fs1[3] = tin * nf13[jloc] + tplus1 * nf13[jplus1] + tplus2 * nf13[jplus2];
  fs1[4] = tin * nf14[jloc] + tplus1 * nf14[jplus1] + tplus2 * nf14[jplus2];
  fs1[5] = tin * nf15[jloc] + tplus1 * nf15[jplus1] + tplus2 * nf15[jplus2];
  fs1[6] = tin * nf16[jloc] + tplus1 * nf16[jplus1] + tplus2 * nf16[jplus2];
  fs1[7] = tin * nf17[jloc] + tplus1 * nf17[jplus1] + tplus2 * nf17[jplus2];
  fs1[8] = tin * nf18[jloc] + tplus1 * nf18[jplus1] + tplus2 * nf18[jplus2];

  fs2[0] = tin * nf20[jloc] + tplus1 * nf20[jplus1] + tplus2 * nf20[jplus2];
  fs2[1] = tin * nf21[jloc] + tplus1 * nf21[jplus1] + tplus2 * nf21[jplus2];
  fs2[2] = tin * nf22[jloc] + tplus1 * nf22[jplus1] + tplus2 * nf22[jplus2];
  fs2[3] = tin * nf23[jloc] + tplus1 * nf23[jplus1] + tplus2 * nf23[jplus2];
  fs2[4] = tin * nf24[jloc] + tplus1 * nf24[jplus1] + tplus2 * nf24[jplus2];
  fs2[5] = tin * nf25[jloc] + tplus1 * nf25[jplus1] + tplus2 * nf25[jplus2];
  fs2[6] = tin * nf26[jloc] + tplus1 * nf26[jplus1] + tplus2 * nf26[jplus2];
  fs2[7] = tin * nf27[jloc] + tplus1 * nf27[jplus1] + tplus2 * nf27[jplus2];
  fs2[8] = tin * nf28[jloc] + tplus1 * nf28[jplus1] + tplus2 * nf28[jplus2];

  ux = tin * uxloc[jloc] + tplus1 * uxloc[jplus1] + tplus2 * uxloc[jplus2];
  uy = tin * uyloc[jloc] + tplus1 * uyloc[jplus1] + tplus2 * uyloc[jplus2];

  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      source =  - ctau1 * ( fs1[index] - neq ) +
        ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      source = - ctau2 * ( fs2[index] - neq ) +
        ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
      break;
    }
  return source;
}

void lw_sources(double nf10[], double nf11[], double nf12[],
                double nf13[], double nf14[], double nf15[],
                double nf16[], double nf17[], double nf18[],
                double nf20[], double nf21[], double nf22[],
                double nf23[], double nf24[], double nf25[],
                double nf26[], double nf27[], double nf28[])
{
  int k;

  for(k=0; k<nnodes_all; k++)
    {
      sf10[k] = nf10[k];
      sf20[k] = nf20[k];
      source10[k] = - ctau1 * (nf10[k] - neq10[k]) -
        ( csforce_x * uxloc[k] + csforce_y * uyloc[k] ) * neq10[k];
      source20[k] = - ctau2 * (nf20[k] - neq20[k]) -
        ( csforce_x * uxloc[k] + csforce_y * uyloc[k] ) * neq20[k];

      switch(boundary_mode[k])
```

```
{
case 0:
  sf11[k] =  t1minus * nf11[nv3[k]] + t1center * nf11[k] +
             t1plus * nf11[nv1[k]];
  source11[k] = lw_sources_bulk(1, 1, nv3[k], k, nv1[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf21[k] =  t2minus * nf21[nv3[k]] + t2center * nf21[k] +
             t2plus * nf21[nv1[k]];
  source21[k] = lw_sources_bulk(2, 1, nv3[k], k, nv1[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf12[k] =  t1minus * nf12[nv4[k]] + t1center * nf12[k] +
             t1plus * nf12[nv2[k]];
  source12[k] = lw_sources_bulk(1, 2, nv4[k], k, nv2[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf22[k] =  t2minus * nf22[nv4[k]] + t2center * nf22[k] +
             t2plus * nf22[nv2[k]];
  source22[k] = lw_sources_bulk(2, 2, nv4[k], k, nv2[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf13[k] =  t1minus * nf13[nv1[k]] + t1center * nf13[k] +
             t1plus * nf13[nv3[k]];
  source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf23[k] =  t2minus * nf23[nv1[k]] + t2center * nf23[k] +
             t2plus * nf23[nv3[k]];
  source23[k] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf14[k] =  t1minus * nf14[nv2[k]] + t1center * nf14[k] +
             t1plus * nf14[nv4[k]];
  source14[k] = lw_sources_bulk(1, 4, nv2[k], k, nv4[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf24[k] =  t2minus * nf24[nv2[k]] + t2center * nf24[k] +
             t2plus * nf24[nv4[k]];
  source24[k] = lw_sources_bulk(2, 4, nv2[k], k, nv4[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf15[k] =  t1minus * nf15[nv7[k]] + t1center * nf15[k] +
             t1plus * nf15[nv5[k]];
  source15[k] = lw_sources_bulk(1, 5, nv7[k], k, nv5[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf25[k] =  t2minus * nf25[nv7[k]] + t2center * nf25[k] +
             t2plus * nf25[nv5[k]];
  source25[k] = lw_sources_bulk(2, 5, nv7[k], k, nv5[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf16[k] =  t1minus * nf16[nv8[k]] + t1center * nf16[k] +
             t1plus * nf16[nv6[k]];
```



```
  source16[k] = lw_sources_bulk(1, 6, nv8[k], k, nv6[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf26[k] =  t2minus * nf26[nv8[k]] + t2center * nf26[k] +
             t2plus * nf26[nv6[k]];
  source26[k] = lw_sources_bulk(2, 6, nv8[k], k, nv6[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf17[k] =  t1minus * nf17[nv5[k]] + t1center * nf17[k] +
             t1plus * nf17[nv7[k]];
  source17[k] = lw_sources_bulk(1, 7, nv5[k], k, nv7[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf27[k] =  t2minus * nf27[nv5[k]] + t2center * nf27[k] +
             t2plus * nf27[nv7[k]];
  source27[k] = lw_sources_bulk(2, 7, nv5[k], k, nv7[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf18[k] =  t1minus * nf18[nv6[k]] + t1center * nf18[k] +
             t1plus * nf18[nv8[k]];
  source18[k] = lw_sources_bulk(1, 8, nv6[k], k, nv8[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf28[k] =  t2minus * nf28[nv6[k]] + t2center * nf28[k] +
             t2plus * nf28[nv8[k]];
  source28[k] = lw_sources_bulk(2, 8, nv6[k], k, nv8[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  break;
case 1:  /*  bottom wall  */
  sf11[k] =  t1minus * nf11[nv3[k]] + t1center * nf11[k] +
             t1plus * nf11[nv1[k]];
  source11[k] = lw_sources_bulk(1, 1, nv3[k], k, nv1[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf21[k] =  t2minus * nf21[nv3[k]] + t2center * nf21[k] +
             t2plus * nf21[nv1[k]];
  source21[k] = lw_sources_bulk(2, 1, nv3[k], k, nv1[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf12[k] =  tin1 * nf12[k] + tin1plus1 * nf12[nv2[k]] +
             tin1plus2 * nf12[nv2[nv2[k]]];
  source12[k] = lw_sources_in(1, 2, k, nv2[k], nv2[nv2[k]],
                tin1, tin1plus1, tin1plus2,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf22[k] =  tin2 * nf22[k] + tin2plus1 * nf22[nv2[k]] +
             tin2plus2 * nf22[nv2[nv2[k]]];
  source22[k] = lw_sources_in(2, 2, k, nv2[k], nv2[nv2[k]],
                tin2, tin2plus1, tin2plus2,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf13[k] =  t1minus * nf13[nv1[k]] + t1center * nf13[k] +
             t1plus * nf13[nv3[k]];
  source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
```

```
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf23[k] =  t2minus * nf23[nv1[k]] + t2center * nf23[k] +
                   t2plus * nf23[nv3[k]];
      source23[k] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf14[k] =  tout1minus2 * nf14[nv2[nv2[k]]] +
                 tout1minus1 * nf14[nv2[k]] +
                       tout1 * nf14[k];
      source14[k] = lw_sources_out(1, 4, nv2[nv2[k]], nv2[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf24[k] =  tout2minus2 * nf24[nv2[nv2[k]]] +
                 tout2minus1 * nf24[nv2[k]] +
                       tout2 * nf24[k];
      source24[k] = lw_sources_out(2, 4, nv2[nv2[k]], nv2[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf15[k] =  tin1 * nf15[k] + tin1plus1 * nf15[nv5[k]] +
                   tin1plus2 * nf15[nv5[nv5[k]]];
      source15[k] = lw_sources_in(1, 5, k, nv5[k], nv5[nv5[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf25[k] =  tin2 * nf25[k] + tin2plus1 * nf25[nv5[k]] +
                   tin2plus2 * nf25[nv5[nv5[k]]];
      source25[k] = lw_sources_in(2, 5, k, nv5[k], nv5[nv5[k]],
                      tin2, tin2plus1, tin2plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf16[k] =  tin1 * nf16[k] + tin1plus1 * nf16[nv6[k]] +
                   tin1plus2 * nf16[nv6[nv6[k]]];
      source16[k] = lw_sources_in(1, 6, k, nv6[k], nv6[nv6[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf26[k] =  tin2 * nf26[k] + tin2plus1 * nf26[nv6[k]] +
                   tin2plus2 * nf26[nv6[nv6[k]]];
      source26[k] = lw_sources_in(2, 6, k, nv6[k], nv6[nv6[k]],
                      tin2, tin2plus1, tin2plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf17[k] =  tout1minus2 * nf17[nv5[nv5[k]]] +
                 tout1minus1 * nf17[nv5[k]] +
                       tout1 * nf17[k];
      source17[k] = lw_sources_out(1, 7, nv5[nv5[k]], nv5[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf27[k] =  tout2minus2 * nf27[nv5[nv5[k]]] +
                 tout2minus1 * nf27[nv5[k]] +
                       tout2 * nf27[k];
      source27[k] = lw_sources_out(2, 7, nv5[nv5[k]], nv5[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf18[k] =  tout1minus2 * nf18[nv6[nv6[k]]] +
                 tout1minus1 * nf18[nv6[k]] +
                       tout1 * nf18[k];
      source18[k] = lw_sources_out(1, 8, nv6[nv6[k]], nv6[k], k,
                      tout1minus2, tout1minus1, tout1,
```



```
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf28[k] =  tout2minus2 * nf28[nv6[nv6[k]]] +
                 tout2minus1 * nf28[nv6[k]] +
                       tout2 * nf28[k];
      source28[k] = lw_sources_out(2, 8, nv6[nv6[k]], nv6[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
/*
      sf11[k] =  t1minus * nf11[nv3[k]] + t1center * nf11[k] +
                   t1plus * nf11[nv1[k]];
      source11[k] = lw_sources_bulk(1, 1, nv3[k], k, nv1[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf21[k] =  t2minus * nf21[nv3[k]] + t2center * nf21[k] +
                   t2plus * nf21[nv1[k]];
      source21[k] = lw_sources_bulk(2, 1, nv3[k], k, nv1[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf12[k] = nf12[k] - cfl1 * (nf12[nv2[k]] - nf12[k]);
      source12[k] = compute_upwind_sources_boundary(1, 2, k, nv2[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf22[k] = nf22[k] - cfl2 * (nf22[nv2[k]] - nf22[k]);
      source22[k] = compute_upwind_sources_boundary(2, 2, k, nv2[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf13[k] =  t1minus * nf13[nv1[k]] + t1center * nf13[k] +
                   t1plus * nf13[nv3[k]];
      source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf23[k] =  t2minus * nf23[nv1[k]] + t2center * nf23[k] +
                   t2plus * nf23[nv3[k]];
      source23[k] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf14[k] = nf14[k] - cfl1 * (nf14[k] - nf14[nv2[k]]);
      source14[k] = compute_upwind_sources_bulk(1, 4, k, nv2[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf24[k] = nf24[k] - cfl2 * (nf24[k] - nf24[nv2[k]]);
      source24[k] = compute_upwind_sources_bulk(2, 4, k, nv2[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf15[k] = nf15[k] - cfl1 * (nf15[nv5[k]] - nf15[k]);
      source15[k] = compute_upwind_sources_boundary(1, 5, k, nv5[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf25[k] = nf25[k] - cfl2 * (nf25[nv5[k]] - nf25[k]);
      source25[k] = compute_upwind_sources_boundary(2, 5, k, nv5[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf16[k] = nf16[k] - cfl1 * (nf16[nv6[k]] - nf16[k]);
      source16[k] = compute_upwind_sources_boundary(1, 6, k, nv6[k],
```

```
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf26[k] = nf26[k] - cf12 * (nf26[nv6[k]] - nf26[k]);
source26[k] = compute_upwind_sources_boundary(2, 6, k, nv6[k],
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]);
source17[k] = compute_upwind_sources_bulk(1, 7, k, nv5[k],
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]);
source27[k] = compute_upwind_sources_bulk(2, 7, k, nv5[k],
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]);
source18[k] = compute_upwind_sources_bulk(1, 8, k, nv6[k],
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf28[k] = nf28[k] - cf12 * (nf28[k] - nf28[nv6[k]]);
source28[k] = compute_upwind_sources_bulk(2, 8, k, nv6[k],
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
                */
/*
sf11[k] =  t1minus * nf11[nv3[k]] + t1center * nf11[k] +
             t1plus * nf11[nv1[k]];
source11[k] = lw_sources_bulk(1, 1, nv3[k], k, nv1[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf21[k] =  t2minus * nf21[nv3[k]] + t2center * nf21[k] +
             t2plus * nf21[nv1[k]];
source21[k] = lw_sources_bulk(2, 1, nv3[k], k, nv1[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf12[k] = nf12[k] - cf11 * (nf12[nv2[k]] - nf12[k]);
source12[k] = compute_upwind_sources_boundary_in(1, 2, k, nv2[k],
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf22[k] = nf22[k] - cf12 * (nf22[nv2[k]] - nf22[k]);
source22[k] = compute_upwind_sources_boundary_in(2, 2, k, nv2[k],
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf13[k] =  t1minus * nf13[nv1[k]] + t1center * nf13[k] +
             t1plus * nf13[nv3[k]];
source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],
                t1minus, t1center, t1plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf23[k] =  t2minus * nf23[nv1[k]] + t2center * nf23[k] +
             t2plus * nf23[nv3[k]];
source23[k] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k],
                t2minus, t2center, t2plus,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]);
source14[k] = compute_upwind_sources_boundary_out(1, 4, nv2[k], k,
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
```



```
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf24[k] = nf24[k] - cf12 * (nf24[k] - nf24[nv2[k]]);
source24[k] = compute_upwind_sources_boundary_out(2, 4, nv2[k], k,
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf15[k] = nf15[k] - cf11 * (nf15[nv5[k]] - nf15[k]);
source15[k] = compute_upwind_sources_boundary_in(1, 5, k, nv5[k],
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf25[k] = nf25[k] - cf12 * (nf25[nv5[k]] - nf25[k]);
source25[k] = compute_upwind_sources_boundary_in(2, 5, k, nv5[k],
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf16[k] = nf16[k] - cf11 * (nf16[nv6[k]] - nf16[k]);
source16[k] = compute_upwind_sources_boundary_in(1, 6, k, nv6[k],
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf26[k] = nf26[k] - cf12 * (nf26[nv6[k]] - nf26[k]);
source26[k] = compute_upwind_sources_boundary_in(2, 6, k, nv6[k],
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]);
source17[k] = compute_upwind_sources_boundary_out(1, 7, nv5[k], k,
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]);
source27[k] = compute_upwind_sources_boundary_out(2, 7, nv5[k], k,
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]);
source18[k] = compute_upwind_sources_boundary_out(1, 8, nv6[k], k,
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf28[k] = nf28[k] - cf12 * (nf28[k] - nf28[nv6[k]]);
source28[k] = compute_upwind_sources_boundary_out(2, 8, nv6[k], k,
                uxwall_bot, uywall_bot,
                nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
*/

  break;
case 2:  /*  top wall  */

  sf11[k] =  t1minus * nf11[nv3[k]] + t1center * nf11[k] +
               t1plus * nf11[nv1[k]];
  source11[k] = lw_sources_bulk(1, 1, nv3[k], k, nv1[k],
                  t1minus, t1center, t1plus,
                  nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                  nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
  sf21[k] =  t2minus * nf21[nv3[k]] + t2center * nf21[k] +
               t2plus * nf21[nv1[k]];
  source21[k] = lw_sources_bulk(2, 1, nv3[k], k, nv1[k],
                  t2minus, t2center, t2plus,
                  nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                  nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

  sf12[k] =  tout1minus2 * nf12[nv4[nv4[k]]] +
```

```
                  tout1minus1 * nf12[nv4[k]] +
                      tout1 * nf12[k];
    source12[k] = lw_sources_out(1, 2, nv4[nv4[k]], nv4[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf22[k] =     tout2minus2 * nf22[nv4[nv4[k]]] +
                  tout2minus1 * nf22[nv4[k]] +
                      tout2 * nf22[k];
    source22[k] = lw_sources_out(2, 2, nv4[nv4[k]], nv4[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf13[k] =     t1minus * nf13[nv1[k]] + t1center * nf13[k] +
                   t1plus * nf13[nv3[k]];
    source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf23[k] =     t2minus * nf23[nv1[k]] + t2center * nf23[k] +
                   t2plus * nf23[nv3[k]];
    source23[k] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf14[k] =     tin1 * nf14[k] + tin1plus1 * nf14[nv4[k]] +
                   tin1plus2 * nf14[nv4[nv4[k]]];
    source14[k] = lw_sources_in(1, 4, k, nv4[k], nv4[nv4[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf24[k] =     tin2 * nf24[k] + tin2plus1 * nf24[nv4[k]] +
                   tin2plus2 * nf24[nv4[nv4[k]]];
    source24[k] = lw_sources_in(2, 4, k, nv4[k], nv4[nv4[k]],
                      tin2, tin2plus1, tin2plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf15[k] =     tout1minus2 * nf15[nv7[nv7[k]]] +
                  tout1minus1 * nf15[nv7[k]] +
                      tout1 * nf15[k];
    source15[k] = lw_sources_out(1, 5, nv7[nv7[k]], nv7[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf25[k] =     tout2minus2 * nf25[nv7[nv7[k]]] +
                  tout2minus1 * nf25[nv7[k]] +
                      tout2 * nf25[k];
    source25[k] = lw_sources_out(2, 5, nv7[nv7[k]], nv7[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf16[k] =     tout1minus2 * nf16[nv8[nv8[k]]] +
                  tout1minus1 * nf16[nv8[k]] +
                      tout1 * nf16[k];
    source16[k] = lw_sources_out(1, 6, nv8[nv8[k]], nv8[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf26[k] =     tout2minus2 * nf26[nv8[nv8[k]]] +
                  tout2minus1 * nf26[nv8[k]] +
                      tout2 * nf26[k];
    source26[k] = lw_sources_out(2, 6, nv8[nv8[k]], nv8[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
```



```
    sf17[k] =     tin1 * nf17[k] + tin1plus1 * nf17[nv7[k]] +
                   tin1plus2 * nf17[nv7[nv7[k]]];
    source17[k] = lw_sources_in(1, 7, k, nv7[k], nv7[nv7[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf27[k] =     tin2 * nf27[k] + tin2plus1 * nf27[nv7[k]] +
                   tin2plus2 * nf27[nv7[nv7[k]]];
    source27[k] = lw_sources_in(2, 7, k, nv7[k], nv7[nv7[k]],
                      tin2, tin2plus1, tin2plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf18[k] =     tin1 * nf18[k] + tin1plus1 * nf18[nv8[k]] +
                   tin1plus2 * nf18[nv8[nv8[k]]];
    source18[k] = lw_sources_in(1, 8, k, nv8[k], nv8[nv8[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf28[k] =     tin2 * nf28[k] + tin2plus1 * nf28[nv8[k]] +
                   tin2plus2 * nf28[nv8[nv8[k]]];
    source28[k] = lw_sources_in(2, 8, k, nv8[k], nv8[nv8[k]],
                      tin2, tin2plus1, tin2plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);


    /*
    sf11[k] =     t1minus * nf11[nv3[k]] + t1center * nf11[k] +
                   t1plus * nf11[nv1[k]];
    source11[k] = lw_sources_bulk(1, 1, nv3[k], k, nv1[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf21[k] =     t2minus * nf21[nv3[k]] + t2center * nf21[k] +
                   t2plus * nf21[nv1[k]];
    source21[k] = lw_sources_bulk(2, 1, nv3[k], k, nv1[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf12[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]);
    source12[k] = compute_upwind_sources_bulk(1, 2, k, nv4[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf22[k] = nf22[k] - cf12 * (nf22[k] - nf22[nv4[k]]);
    source22[k] = compute_upwind_sources_bulk(2, 2, k, nv4[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf13[k] =     t1minus * nf13[nv1[k]] + t1center * nf13[k] +
                   t1plus * nf13[nv3[k]];
    source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf23[k] =     t2minus * nf23[nv1[k]] + t2center * nf23[k] +
                   t2plus * nf23[nv3[k]];
    source23[k] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf14[k] = nf14[k] - cf11 * (nf14[nv4[k]] - nf14[k]);
    source14[k] = compute_upwind_sources_boundary(1, 4, k, nv4[k],
                      uxwall_top, uywall_top,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
```

```
sf24[k] = nf24[k] - cfl2 * (nf24[nv4[k]] - nf24[k]);
source24[k] = compute_upwind_sources_boundary(2, 4, k, nv4[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf15[k] = nf15[k] - cfl1 * (nf15[k] - nf15[nv7[k]]);
source15[k] = compute_upwind_sources_bulk(1, 5, k, nv7[k],
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf25[k] = nf25[k] - cfl2 * (nf25[k] - nf25[nv7[k]]);
source25[k] = compute_upwind_sources_bulk(2, 5, k, nv7[k],
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf16[k] = nf16[k] - cfl1 * (nf16[k] - nf16[nv8[k]]);
source16[k] = compute_upwind_sources_bulk(1, 6, k, nv8[k],
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf26[k] = nf26[k] - cfl2 * (nf26[k] - nf26[nv8[k]]);
source26[k] = compute_upwind_sources_bulk(2, 6, k, nv8[k],
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf17[k] = nf17[k] - cfl1 * (nf17[nv7[k]] - nf17[k]);
source17[k] = compute_upwind_sources_boundary(1, 7, k, nv7[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf27[k] = nf27[k] - cfl2 * (nf27[nv7[k]] - nf27[k]);
source27[k] = compute_upwind_sources_boundary(2, 7, k, nv7[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf18[k] = nf18[k] - cfl1 * (nf18[nv8[k]] - nf18[k]);
source18[k] = compute_upwind_sources_boundary(1, 8, k, nv8[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf28[k] = nf28[k] - cfl2 * (nf28[nv8[k]] - nf28[k]);
source28[k] = compute_upwind_sources_boundary(2, 8, k, nv8[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

*/
/*
sf11[k] =  t1minus * nf11[nv3[k]] + t1center * nf11[k] +
           t1plus * nf11[nv1[k]];
source11[k] = lw_sources_bulk(1, 1, nv3[k], k, nv1[k],
               t1minus, t1center, t1plus,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf21[k] =  t2minus * nf21[nv3[k]] + t2center * nf21[k] +
           t2plus * nf21[nv1[k]];
source21[k] = lw_sources_bulk(2, 1, nv3[k], k, nv1[k],
               t2minus, t2center, t2plus,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf12[k] = nf12[k] - cfl1 * (nf12[k] - nf12[nv4[k]]);
source12[k] = compute_upwind_sources_boundary_out(1, 2, nv4[k], k,
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf22[k] = nf22[k] - cfl2 * (nf22[k] - nf22[nv4[k]]);
source22[k] = compute_upwind_sources_boundary_out(2, 2, nv4[k], k,
               uxwall_top, uywall_top,
```



```
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf13[k] =  t1minus * nf13[nv1[k]] + t1center * nf13[k] +
           t1plus * nf13[nv3[k]];
source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],
               t1minus, t1center, t1plus,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf23[k] =  t2minus * nf23[nv1[k]] + t2center * nf23[k] +
           t2plus * nf23[nv3[k]];
source23[k] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k],
               t2minus, t2center, t2plus,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf14[k] = nf14[k] - cfl1 * (nf14[nv4[k]] - nf14[k]);
source14[k] = compute_upwind_sources_boundary_in(1, 4, k, nv4[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf24[k] = nf24[k] - cfl2 * (nf24[nv4[k]] - nf24[k]);
source24[k] = compute_upwind_sources_boundary_in(2, 4, k, nv4[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf15[k] = nf15[k] - cfl1 * (nf15[k] - nf15[nv7[k]]);
source15[k] = compute_upwind_sources_boundary_out(1, 5, nv7[k], k,
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf25[k] = nf25[k] - cfl2 * (nf25[k] - nf25[nv7[k]]);
source25[k] = compute_upwind_sources_boundary_out(2, 5, nv7[k], k,
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf16[k] = nf16[k] - cfl1 * (nf16[k] - nf16[nv8[k]]);
source16[k] = compute_upwind_sources_boundary_out(1, 6, nv8[k], k,
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf26[k] = nf26[k] - cfl2 * (nf26[k] - nf26[nv8[k]]);
source26[k] = compute_upwind_sources_boundary_out(2, 6, nv8[k], k,
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf17[k] = nf17[k] - cfl1 * (nf17[nv7[k]] - nf17[k]);
source17[k] = compute_upwind_sources_boundary_in(1, 7, k, nv7[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf27[k] = nf27[k] - cfl2 * (nf27[nv7[k]] - nf27[k]);
source27[k] = compute_upwind_sources_boundary_in(2, 7, k, nv7[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

sf18[k] = nf18[k] - cfl1 * (nf18[nv8[k]] - nf18[k]);
source18[k] = compute_upwind_sources_boundary_in(1, 8, k, nv8[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
               nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf28[k] = nf28[k] - cfl2 * (nf28[nv8[k]] - nf28[k]);
source28[k] = compute_upwind_sources_boundary_in(2, 8, k, nv8[k],
               uxwall_top, uywall_top,
               nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
```

```
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       */
       break;
     case 3:
       sf11[k] =  tin1 * nf11[k] + tin1plus1 * nf11[nv1[k]] +
                                tin1plus2 * nf11[nv1[nv1[k]]];
       source11[k] = lw_sources_in(1, 1, k, nv1[k], nv1[nv1[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf21[k] =  tin2 * nf21[k] + tin2plus1 * nf21[nv1[k]] +
                                tin2plus2 * nf21[nv1[nv1[k]]];
       source21[k] = lw_sources_in(2, 1, k, nv1[k], nv1[nv1[k]],
                      tin2, tin2plus1, tin2plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

       sf12[k] =  t1minus * nf12[nv4[k]] + t1center * nf12[k] +
                   t1plus * nf12[nv2[k]];
       source12[k] = lw_sources_bulk(1, 2, nv4[k], k, nv2[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf22[k] =  t2minus * nf22[nv4[k]] + t2center * nf22[k] +
                   t2plus * nf22[nv2[k]];
       source22[k] = lw_sources_bulk(2, 2, nv4[k], k, nv2[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);


       sf13[k] =  tout1minus2 * nf13[nv1[nv1[k]]] +
                  tout1minus1 * nf13[nv1[k]] +
                        tout1 * nf13[k];
       source13[k] = lw_sources_out(1, 3, nv1[nv1[k]], nv1[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf23[k] =  tout2minus2 * nf23[nv1[nv1[k]]] +
                  tout2minus1 * nf23[nv1[k]] +
                        tout2 * nf23[k];
       source23[k] = lw_sources_out(2, 3, nv1[nv1[k]], nv1[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

       sf14[k] =  t1minus * nf14[nv2[k]] + t1center * nf14[k] +
                   t1plus * nf14[nv4[k]];
       source14[k] = lw_sources_bulk(1, 4, nv2[k], k, nv4[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf24[k] =  t2minus * nf24[nv2[k]] + t2center * nf24[k] +
                   t2plus * nf24[nv4[k]];
       source24[k] = lw_sources_bulk(2, 4, nv2[k], k, nv4[k],
                      t2minus, t2center, t2plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

       sf15[k] =  tin1 * nf15[k] + tin1plus1 * nf15[nv5[k]] +
                   tin1plus2 * nf15[nv5[nv5[k]]];
       source15[k] = lw_sources_in(1, 5, k, nv5[k], nv5[nv5[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf25[k] =  tin2 * nf25[k] + tin2plus1 * nf25[nv5[k]] +
                   tin2plus2 * nf25[nv5[nv5[k]]];
       source25[k] = lw_sources_in(2, 5, k, nv5[k], nv5[nv5[k]],
                      tin2, tin2plus1, tin2plus2,
```



```
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

       sf16[k] =  tout1minus2 * nf16[nv8[nv8[k]]] +
                  tout1minus1 * nf16[nv8[k]] +
                        tout1 * nf16[k];
       source16[k] = lw_sources_out(1, 6, nv8[nv8[k]], nv8[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf26[k] =  tout2minus2 * nf26[nv8[nv8[k]]] +
                  tout2minus1 * nf26[nv8[k]] +
                        tout2 * nf26[k];
       source26[k] = lw_sources_out(2, 6, nv8[nv8[k]], nv8[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

       sf17[k] =  tout1minus2 * nf17[nv5[nv5[k]]] +
                  tout1minus1 * nf17[nv5[k]] +
                        tout1 * nf17[k];
       source17[k] = lw_sources_out(1, 7, nv5[nv5[k]], nv5[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf27[k] =  tout2minus2 * nf27[nv5[nv5[k]]] +
                  tout2minus1 * nf27[nv5[k]] +
                        tout2 * nf27[k];
       source27[k] = lw_sources_out(2, 7, nv5[nv5[k]], nv5[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

       sf18[k] =  tin1 * nf18[k] + tin1plus1 * nf18[nv8[k]] +
                                tin1plus2 * nf18[nv8[nv8[k]]];
       source18[k] = lw_sources_in(1, 8, k, nv8[k], nv8[nv8[k]],
                      tin1, tin1plus1, tin1plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf28[k] =  tin2 * nf28[k] + tin2plus1 * nf28[nv8[k]] +
                                tin2plus2 * nf28[nv8[nv8[k]]];
       source28[k] = lw_sources_in(2, 8, k, nv8[k], nv8[nv8[k]],
                      tin2, tin2plus1, tin2plus2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       break;
     case 4:
       sf11[k] =  tout1minus2 * nf11[nv3[nv3[k]]] +
                  tout1minus1 * nf11[nv3[k]] +
                        tout1 * nf11[k];
       source11[k] = lw_sources_out(1, 1, nv3[nv3[k]], nv3[k], k,
                      tout1minus2, tout1minus1, tout1,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf21[k] =  tout2minus2 * nf21[nv3[nv3[k]]] +
                  tout2minus1 * nf21[nv3[k]] +
                        tout2 * nf21[k];
       source21[k] = lw_sources_out(2, 1, nv3[nv3[k]], nv3[k], k,
                      tout2minus2, tout2minus1, tout2,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

       sf12[k] =  t1minus * nf12[nv4[k]] + t1center * nf12[k] +
                   t1plus * nf12[nv2[k]];
       source12[k] = lw_sources_bulk(1, 2, nv4[k], k, nv2[k],
                      t1minus, t1center, t1plus,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
       sf22[k] =  t2minus * nf22[nv4[k]] + t2center * nf22[k] +
```

```
            t2plus * nf22[nv2[k]];
      source22[k] = lw_sources_bulk(2, 2, nv4[k], k, nv2[k],
                    t2minus, t2center, t2plus,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf13[k] =  tin1 * nf13[k] + tin1plus1 * nf13[nv3[k]] +
                             tin1plus2 * nf13[nv3[nv3[k]]];
      source13[k] = lw_sources_in(1, 3, k, nv3[k], nv3[nv3[k]],
                    tin1, tin1plus1, tin1plus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf23[k] =  tin2 * nf23[k] + tin2plus1 * nf23[nv3[k]] +
                             tin2plus2 * nf23[nv3[nv3[k]]];
      source23[k] = lw_sources_in(2, 3, k, nv3[k], nv3[nv3[k]],
                    tin2, tin2plus1, tin2plus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf14[k] =  t1minus * nf14[nv2[k]] + t1center * nf14[k] +
                 t1plus * nf14[nv4[k]];
      source14[k] = lw_sources_bulk(1, 4, nv2[k], k, nv4[k],
                    t1minus, t1center, t1plus,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf24[k] =  t2minus * nf24[nv2[k]] + t2center * nf24[k] +
                 t2plus * nf24[nv4[k]];
      source24[k] = lw_sources_bulk(2, 4, nv2[k], k, nv4[k],
                    t2minus, t2center, t2plus,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf15[k] =  tout1minus2 * nf15[nv7[nv7[k]]] +
                 tout1minus1 * nf15[nv7[k]] +
                      tout1 * nf15[k];
      source15[k] = lw_sources_out(1, 5, nv7[nv7[k]], nv7[k], k,
                    tout1minus2, tout1minus1, tout1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf25[k] =  tout2minus2 * nf25[nv7[nv7[k]]] +
                 tout2minus1 * nf25[nv7[k]] +
                      tout2 * nf25[k];
      source25[k] = lw_sources_out(2, 5, nv7[nv7[k]], nv7[k], k,
                    tout2minus2, tout2minus1, tout2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf16[k] =  tin1 * nf16[k] + tin1plus1 * nf16[nv6[k]] +
                             tin1plus2 * nf16[nv6[nv6[k]]];
      source16[k] = lw_sources_in(1, 6, k, nv6[k], nv6[nv6[k]],
                    tin1, tin1plus1, tin1plus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf26[k] =  tin2 * nf26[k] + tin2plus1 * nf26[nv6[k]] +
                             tin2plus2 * nf26[nv6[nv6[k]]];
      source26[k] = lw_sources_in(2, 6, k, nv6[k], nv6[nv6[k]],
                    tin2, tin2plus1, tin2plus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf17[k] =  tin1 * nf17[k] + tin1plus1 * nf17[nv7[k]] +
                             tin1plus2 * nf17[nv7[nv7[k]]];
      source17[k] = lw_sources_in(1, 7, k, nv7[k], nv7[nv7[k]],
                    tin1, tin1plus1, tin1plus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf27[k] =  tin2 * nf27[k] + tin2plus1 * nf27[nv7[k]] +
                             tin2plus2 * nf27[nv7[nv7[k]]];
      source27[k] = lw_sources_in(2, 7, k, nv7[k], nv7[nv7[k]],
```



```
                    tin2, tin2plus1, tin2plus2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      sf18[k] =  tout1minus2 * nf18[nv6[nv6[k]]] +
                 tout1minus1 * nf18[nv6[k]] +
                      tout1 * nf18[k];
      source18[k] = lw_sources_out(1, 8, nv6[nv6[k]], nv6[k], k,
                    tout1minus2, tout1minus1, tout1,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
      sf28[k] =  tout2minus2 * nf28[nv6[nv6[k]]] +
                 tout2minus1 * nf28[nv6[k]] +
                      tout2 * nf28[k];
      source28[k] = lw_sources_out(2, 8, nv6[nv6[k]], nv6[k], k,
                    tout2minus2, tout2minus1, tout2,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

      break;
     }
   }
}

void lw(double sf0[], double sf1[], double sf2[],
        double sf3[], double sf4[], double sf5[],
        double sf6[], double sf7[], double sf8[],
        double source0[], double source1[], double source2[],
        double source3[], double source4[], double source5[],
        double source6[], double source7[], double source8[],
        double nff0[], double nff1[], double nff2[],
        double nff3[], double nff4[], double nff5[],
        double nff6[], double nff7[], double nff8[])
{
  int k;

  for(k=0; k<nnodes_all; k++)
    {
     nff0[k] = sf0[k] + source0[k];
     nff1[k] = sf1[k] + source1[k];
     nff2[k] = sf2[k] + source2[k];
     nff3[k] = sf3[k] + source3[k];
     nff4[k] = sf4[k] + source4[k];
     nff5[k] = sf5[k] + source5[k];
     nff6[k] = sf6[k] + source6[k];
     nff7[k] = sf7[k] + source7[k];
     nff8[k] = sf8[k] + source8[k];
    }
}
```

```c
/**********************************************************************\
*                                                                      *
*   wet9fd.c                                                           *
*                                                                      *
\**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "wet9head.h"

void first_upwind(double mass, double tau, double cspeed,
                  double nf0[], double nf1[], double nf2[],
                  double nf3[], double nf4[], double nf5[],
                  double nf6[], double nf7[], double nf8[],
                  double nff0[], double nff1[], double nff2[],
                  double nff3[], double nff4[], double nff5[],
                  double nff6[], double nff7[], double nff8[],
                  double neq0[], double neq1[], double neq2[],
                  double neq3[], double neq4[], double neq5[],
                  double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxs2, cgradys2, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_x;
  cgradys2 = cspeed * sqrt((double) 2) * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                 (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));


  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */
          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
            - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
            - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
            - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
            - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
          /*
          printf("k==%d     %g  %g  %g  %g  %g  %g  %g  %g  %g  %g  %g\n",k,
```



```c
                 neq0[k],neq1[k],neq2[k],neq3[k],neq4[k],neq5[k],
                 neq6[k],neq7[k],neq8[k],uxloc[k],uyloc[k]);
          printf("k==%d   nf %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
                 k,nf0[k],nf1[k],nf2[k],nf3[k],nf4[k],nf5[k],
                 nf6[k],nf7[k],nf8[k]);
          printf("k==%d  nff %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
                 k,nff0[k],nff1[k],nff2[k],nff3[k],nff4[k],nff5[k],
                 nff6[k],nff7[k],nff8[k]);
          */
          break;

        case 3:   /*  left  wall  */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx * ecx[1] * (nf1[k] - nf1[k]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx * ecx[5] * (nf5[k] - nf5[k])
            - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
            - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
            - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx * ecx[8] * (nf8[k] - nf8[k])
            - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
          break;

        case 4:   /*  right  wall */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx * ecx[3] * (nf3[k] - nf3[k]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
            - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx * ecx[6] * (nf6[k] - nf6[k])
            - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx * ecx[7] * (nf7[k] - nf7[k])
            - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
            - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
          break;
        }
    }
}
```

```c
/*********************************************************************\
*                                                                     *
*   wet9is.c                                                          *
*                                                                     *
\*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "wet9head.h"


void islb(void)
{
  int k;
  double prodscal;
  for(k=0; k<nnodes_all; k++)
    {
     prodscal = csforce_x * uxloc[k] + csforce_y * uxloc[k];

     ff10[k]  =  f10[k]-ctau1*(f10[k]-neq10[k])+(ecprod1[0]-prodscal)*neq10[k];
     ff11[k]  =  f11[k]-ctau1*(f11[k]-neq11[k])+(ecprod1[1]-prodscal)*neq11[k];
     ff12[k]  =  f12[k]-ctau1*(f12[k]-neq12[k])+(ecprod1[2]-prodscal)*neq12[k];
     ff13[k]  =  f13[k]-ctau1*(f13[k]-neq13[k])+(ecprod1[3]-prodscal)*neq13[k];
     ff14[k]  =  f14[k]-ctau1*(f14[k]-neq14[k])+(ecprod1[4]-prodscal)*neq14[k];
     ff15[k]  =  f15[k]-ctau1*(f15[k]-neq15[k])+(ecprod1[5]-prodscal)*neq15[k];
     ff16[k]  =  f16[k]-ctau1*(f16[k]-neq16[k])+(ecprod1[6]-prodscal)*neq16[k];
     ff17[k]  =  f17[k]-ctau1*(f17[k]-neq17[k])+(ecprod1[7]-prodscal)*neq17[k];
     ff18[k]  =  f18[k]-ctau1*(f18[k]-neq18[k])+(ecprod1[8]-prodscal)*neq18[k];

     ff20[k]  =  f20[k]-ctau2*(f20[k]-neq20[k])+(ecprod2[0]-prodscal)*neq20[k];
     ff21[k]  =  f21[k]-ctau2*(f21[k]-neq21[k])+(ecprod2[1]-prodscal)*neq21[k];
     ff22[k]  =  f22[k]-ctau2*(f22[k]-neq22[k])+(ecprod2[2]-prodscal)*neq22[k];
     ff23[k]  =  f23[k]-ctau2*(f23[k]-neq23[k])+(ecprod2[3]-prodscal)*neq23[k];
     ff24[k]  =  f24[k]-ctau2*(f24[k]-neq24[k])+(ecprod2[4]-prodscal)*neq24[k];
     ff25[k]  =  f25[k]-ctau2*(f25[k]-neq25[k])+(ecprod2[5]-prodscal)*neq25[k];
     ff26[k]  =  f26[k]-ctau2*(f26[k]-neq26[k])+(ecprod2[6]-prodscal)*neq26[k];
     ff27[k]  =  f27[k]-ctau2*(f27[k]-neq27[k])+(ecprod2[7]-prodscal)*neq27[k];
     ff28[k]  =  f28[k]-ctau2*(f28[k]-neq28[k])+(ecprod2[8]-prodscal)*neq28[k];
    }

  for(k=0; k<nnodes_all; k++)
    {
     switch(boundary_mode[k])
       {
        case 0:

          f10[k] = ff10[k];
          f11[k] = isminus1 * ff11[nv3[k]] + iscenter1 * ff11[k] +
                   isplus1 * ff11[nv1[k]];
          f12[k] = isminus1 * ff12[nv4[k]] + iscenter1 * ff12[k] +
                   isplus1 * ff12[nv2[k]];
          f13[k] = isminus1 * ff13[nv1[k]] + iscenter1 * ff13[k] +
                   isplus1 * ff13[nv3[k]];
          f14[k] = isminus1 * ff14[nv2[k]] + iscenter1 * ff14[k] +
                   isplus1 * ff14[nv4[k]];
          f15[k] = isminus1 * ff15[nv7[k]] + iscenter1 * ff15[k] +
                   isplus1 * ff15[nv5[k]];
          f16[k] = isminus1 * ff16[nv8[k]] + iscenter1 * ff16[k] +
                   isplus1 * ff16[nv6[k]];
          f17[k] = isminus1 * ff17[nv5[k]] + iscenter1 * ff17[k] +
                   isplus1 * ff17[nv7[k]];
          f18[k] = isminus1 * ff18[nv6[k]] + iscenter1 * ff18[k] +
                   isplus1 * ff18[nv8[k]];

          f20[k] = ff20[k];
          f21[k] = isminus2 * ff21[nv3[k]] + iscenter2 * ff21[k] +
                   isplus2 * ff21[nv1[k]];
```



```c
          f22[k] = isminus2 * ff22[nv4[k]] + iscenter2 * ff22[k] +
                   isplus2 * ff22[nv2[k]];
          f23[k] = isminus2 * ff23[nv1[k]] + iscenter2 * ff23[k] +
                   isplus2 * ff23[nv3[k]];
          f24[k] = isminus2 * ff24[nv2[k]] + iscenter2 * ff24[k] +
                   isplus2 * ff24[nv4[k]];
          f25[k] = isminus2 * ff25[nv7[k]] + iscenter2 * ff25[k] +
                   isplus2 * ff25[nv5[k]];
          f26[k] = isminus2 * ff26[nv8[k]] + iscenter2 * ff26[k] +
                   isplus2 * ff26[nv6[k]];
          f27[k] = isminus2 * ff27[nv5[k]] + iscenter2 * ff27[k] +
                   isplus2 * ff27[nv7[k]];
          f28[k] = isminus2 * ff28[nv6[k]] + iscenter2 * ff28[k] +
                   isplus2 * ff28[nv8[k]];
          break;


        case 1:

          f10[k] = ff10[k];
          f11[k] = isminus1 * ff11[nv3[k]] + iscenter1 * ff11[k] +
                   isplus1 * ff11[nv1[k]];
          f12[k] = tin1 * ff12[k] + tin1plus1 * ff12[nv2[k]] +
                                tin1plus2 * ff12[nv2[nv2[k]]];
          f13[k] = isminus1 * ff13[nv1[k]] + iscenter1 * ff13[k] +
                   isplus1 * ff13[nv3[k]];
          f14[k] = tout1minus2 * ff14[nv2[nv2[k]]] +
                   tout1minus1 * ff14[nv2[k]] + tout1 * ff14[k];
          f15[k] = tin1 * ff15[k] + tin1plus1 * ff15[nv5[k]] +
                                tin1plus2 * ff15[nv5[nv5[k]]];
          f16[k] = tin1 * ff16[k] + tin1plus1 * ff16[nv6[k]] +
                                tin1plus2 * ff16[nv6[nv6[k]]];
          f17[k] = tout1minus2 * ff17[nv5[nv5[k]]] +
                   tout1minus1 * ff17[nv5[k]] + tout1 * ff17[k];
          f18[k] = tout1minus2 * ff18[nv6[nv6[k]]] +
                   tout1minus1 * ff18[nv6[k]] + tout1 * ff18[k];

          f20[k] = ff20[k];
          f21[k] = isminus2 * ff21[nv3[k]] + iscenter2 * ff21[k] +
                   isplus2 * ff21[nv1[k]];
          f22[k] = tin2 * ff22[k] + tin2plus1 * ff22[nv2[k]] +
                                tin2plus2 * ff22[nv2[nv2[k]]];
          f23[k] = isminus2 * ff23[nv1[k]] + iscenter2 * ff23[k] +
                   isplus2 * ff23[nv3[k]];
          f24[k] = tout2minus2 * ff24[nv2[nv2[k]]] +
                   tout2minus1 * ff24[nv2[k]] + tout2 * ff24[k];
          f25[k] = tin2 * ff25[k] + tin2plus1 * ff25[nv5[k]] +
                                tin2plus2 * ff25[nv5[nv5[k]]];
          f26[k] = tin2 * ff26[k] + tin2plus1 * ff26[nv6[k]] +
                                tin2plus2 * ff26[nv6[nv6[k]]];
          f27[k] = tout2minus2 * ff27[nv5[nv5[k]]] +
                   tout2minus1 * ff27[nv5[k]] + tout2 * ff27[k];
          f28[k] = tout2minus2 * ff28[nv6[nv6[k]]] +
                   tout2minus1 * ff28[nv6[k]] + tout2 * ff28[k];

          break;
        case 2:

          f10[k] = ff10[k];
          f11[k] = isminus1 * ff11[nv3[k]] + iscenter1 * ff11[k] +
                   isplus1 * ff11[nv1[k]];
          f12[k] = tout1minus2 * ff12[nv4[nv4[k]]] +
                   tout1minus1 * ff12[nv4[k]] + tout1 * ff12[k];
          f13[k] = isminus1 * ff13[nv1[k]] + iscenter1 * ff13[k] +
                   isplus1 * ff13[nv3[k]];
          f14[k] = tin1 * ff14[k] + tin1plus1 * ff14[nv4[k]] +
                                tin1plus2 * ff14[nv4[nv4[k]]];
          f15[k] = tout1minus2 * ff15[nv7[nv7[k]]] +
                   tout1minus1 * ff15[nv7[k]] + tout1 * ff15[k];
```

```
        f16[k] = tout1minus2 * ff16[nv8[nv8[k]]] +
                 tout1minus1 * ff16[nv8[k]] + tout1 * ff16[k];
        f17[k] = tin1 * ff17[k] + tin1plus1 * ff17[nv7[k]] +
                                  tin1plus2 * ff17[nv7[nv7[k]]];
        f18[k] = tin1 * ff18[k] + tin1plus1 * ff18[nv8[k]] +
                                  tin1plus2 * ff18[nv8[nv8[k]]];


        f20[k] = ff20[k];
        f21[k] = isminus2 * ff21[nv3[k]] + iscenter2 * ff21[k] +
                  isplus2 * ff21[nv1[k]];
        f22[k] = tout2minus2 * ff22[nv4[nv4[k]]] +
                 tout2minus1 * ff22[nv4[k]] + tout2 * ff22[k];
        f23[k] = isminus2 * ff23[nv1[k]] + iscenter2 * ff23[k] +
                  isplus2 * ff23[nv3[k]];
        f24[k] = tin2 * ff24[k] + tin2plus1 * ff24[nv4[k]] +
                                  tin2plus2 * ff24[nv4[nv4[k]]];
        f25[k] = tout2minus2 * ff25[nv7[nv7[k]]] +
                 tout2minus1 * ff25[nv7[k]] + tout2 * ff25[k];
        f26[k] = tout2minus2 * ff26[nv8[nv8[k]]] +
                 tout2minus1 * ff26[nv8[k]] + tout2 * ff26[k];
        f27[k] = tin2 * ff27[k] + tin2plus1 * ff27[nv7[k]] +
                                  tin2plus2 * ff27[nv7[nv7[k]]];
        f28[k] = tin2 * ff28[k] + tin2plus1 * ff28[nv8[k]] +
                                  tin2plus2 * ff28[nv8[nv8[k]]];

        break;

      case 3:

        f10[k] = ff10[k];
        f11[k] = tin1 * ff11[k] + tin1plus1 * ff11[nv1[k]] +
                                  tin1plus2 * ff11[nv1[nv1[k]]];
        f12[k] = isminus1 * ff12[nv4[k]] + iscenter1 * ff12[k] +
                  isplus1 * ff12[nv2[k]];
        f13[k] = tout1minus2 * ff13[nv1[nv1[k]]] +
                 tout1minus1 * ff13[nv1[k]] + tout1 * ff13[k];
        f14[k] = isminus1 * ff14[nv2[k]] + iscenter1 * ff14[k] +
                  isplus1 * ff14[nv4[k]];
        f15[k] = tin1 * ff15[k] + tin1plus1 * ff15[nv5[k]] +
                                  tin1plus2 * ff15[nv5[nv5[k]]];
        f16[k] = tout1minus2 * ff16[nv8[nv8[k]]] +
                 tout1minus1 * ff16[nv8[k]] + tout1 * ff16[k];
        f17[k] = tout1minus2 * ff17[nv5[nv5[k]]] +
                 tout1minus1 * ff17[nv5[k]] + tout1 * ff17[k];
        f18[k] = tin1 * ff18[k] + tin1plus1 * ff18[nv8[k]] +
                                  tin1plus2 * ff18[nv8[nv8[k]]];

        f20[k] = ff20[k];
        f21[k] = tin1 * ff21[k] + tin1plus1 * ff21[nv1[k]] +
                                  tin1plus2 * ff21[nv1[nv1[k]]];
        f22[k] = isminus1 * ff22[nv4[k]] + iscenter1 * ff22[k] +
                  isplus1 * ff22[nv2[k]];
        f23[k] = tout1minus2 * ff23[nv1[nv1[k]]] +
                 tout1minus1 * ff23[nv1[k]] + tout1 * ff23[k];
        f24[k] = isminus1 * ff24[nv2[k]] + iscenter1 * ff24[k] +
                  isplus1 * ff24[nv4[k]];
        f25[k] = tin1 * ff25[k] + tin1plus1 * ff25[nv5[k]] +
                                  tin1plus2 * ff25[nv5[nv5[k]]];
        f26[k] = tout1minus2 * ff26[nv8[nv8[k]]] +
                 tout1minus1 * ff26[nv8[k]] + tout1 * ff26[k];
        f27[k] = tout1minus2 * ff27[nv5[nv5[k]]] +
                 tout1minus1 * ff27[nv5[k]] + tout1 * ff27[k];
        f28[k] = tin1 * ff28[k] + tin1plus1 * ff28[nv8[k]] +
                                  tin1plus2 * ff28[nv8[nv8[k]]];

        break;

      case 4:
```



```
        f10[k] = ff10[k];
        f11[k] = tout1minus2 * ff11[nv3[nv3[k]]] +
                 tout1minus1 * ff11[nv3[k]] + tout1 * ff11[k];
        f12[k] = isminus1 * ff12[nv4[k]] + iscenter1 * ff12[k] +
                  isplus1 * ff12[nv2[k]];
        f13[k] = tin1 * ff13[k] + tin1plus1 * ff13[nv3[k]] +
                                  tin1plus2 * ff13[nv3[nv3[k]]];
        f14[k] = isminus1 * ff14[nv2[k]] + iscenter1 * ff14[k] +
                  isplus1 * ff14[nv4[k]];
        f15[k] = tout1minus2 * ff15[nv7[nv7[k]]] +
                 tout1minus1 * ff15[nv7[k]] + tout1 * ff15[k];
        f16[k] = tin1 * ff16[k] + tin1plus1 * ff16[nv6[k]] +
                                  tin1plus2 * ff16[nv6[nv6[k]]];
        f17[k] = tin1 * ff17[k] + tin1plus1 * ff17[nv7[k]] +
                                  tin1plus2 * ff17[nv7[nv7[k]]];
        f18[k] = tout1minus2 * ff18[nv6[nv6[k]]] +
                 tout1minus1 * ff18[nv6[k]] + tout1 * ff18[k];

        f20[k] = ff20[k];
        f21[k] = tout1minus2 * ff21[nv3[nv3[k]]] +
                 tout1minus1 * ff21[nv3[k]] + tout1 * ff21[k];
        f22[k] = isminus1 * ff22[nv4[k]] + iscenter1 * ff22[k] +
                  isplus1 * ff22[nv2[k]];
        f23[k] = tin1 * ff23[k] + tin1plus1 * ff23[nv3[k]] +
                                  tin1plus2 * ff23[nv3[nv3[k]]];
        f24[k] = isminus1 * ff24[nv2[k]] + iscenter1 * ff24[k] +
                  isplus1 * ff24[nv4[k]];
        f25[k] = tout1minus2 * ff25[nv7[nv7[k]]] +
                 tout1minus1 * ff25[nv7[k]] + tout1 * ff25[k];
        f26[k] = tin1 * ff26[k] + tin1plus1 * ff26[nv6[k]] +
                                  tin1plus2 * ff26[nv6[nv6[k]]];
        f27[k] = tin1 * ff27[k] + tin1plus1 * ff27[nv7[k]] +
                                  tin1plus2 * ff27[nv7[nv7[k]]];
        f28[k] = tout1minus2 * ff28[nv6[nv6[k]]] +
                 tout1minus1 * ff28[nv6[k]] + tout1 * ff28[k];

        break;
      }
    }
}

double cent1(double n1, double n2, double n3)
{
  return (isminus1 * n1 + iscenter1 * n2 + isplus1 * n3);
}

double cent2(double n1, double n2, double n3)
{
  return (isminus2 * n1 + iscenter2 * n2 + isplus2 * n3);
}

double fin1(double n1, double n2, double n3)
{
  return (tin1 * n1 + tin1plus1 * n2 + tin1plus2 * n3);
}

double fin2(double n1, double n2, double n3)
{
  return (tin2 * n1 + tin2plus1 * n2 + tin2plus2 * n3);
}

double out1(double n1, double n2, double n3)
{
  return (up1minus2 * n1 + up1minus1 * n2 + up1 * n3);
}

double out2(double n1, double n2, double n3)
{
```

```c
  return (up2minus2 * n1 + up2minus1 * n2 + up2 * n3);
}

void islb_upwind(void)
{
  int k;
  double prodscal;
  for(k=0; k<nnodes_all; k++)
    {
      prodscal = csforce_x * uxloc[k] + csforce_y * uxloc[k];

      ff10[k] =  f10[k]-ctau1*(f10[k]-neq10[k])+(ecprod1[0]-prodscal)*neq10[k];
      ff11[k] =  f11[k]-ctau1*(f11[k]-neq11[k])+(ecprod1[1]-prodscal)*neq11[k];
      ff12[k] =  f12[k]-ctau1*(f12[k]-neq12[k])+(ecprod1[2]-prodscal)*neq12[k];
      ff13[k] =  f13[k]-ctau1*(f13[k]-neq13[k])+(ecprod1[3]-prodscal)*neq13[k];
      ff14[k] =  f14[k]-ctau1*(f14[k]-neq14[k])+(ecprod1[4]-prodscal)*neq14[k];
      ff15[k] =  f15[k]-ctau1*(f15[k]-neq15[k])+(ecprod1[5]-prodscal)*neq15[k];
      ff16[k] =  f16[k]-ctau1*(f16[k]-neq16[k])+(ecprod1[6]-prodscal)*neq16[k];
      ff17[k] =  f17[k]-ctau1*(f17[k]-neq17[k])+(ecprod1[7]-prodscal)*neq17[k];
      ff18[k] =  f18[k]-ctau1*(f18[k]-neq18[k])+(ecprod1[8]-prodscal)*neq18[k];

      ff20[k] =  f20[k]-ctau2*(f20[k]-neq20[k])+(ecprod2[0]-prodscal)*neq20[k];
      ff21[k] =  f21[k]-ctau2*(f21[k]-neq21[k])+(ecprod2[1]-prodscal)*neq21[k];
      ff22[k] =  f22[k]-ctau2*(f22[k]-neq22[k])+(ecprod2[2]-prodscal)*neq22[k];
      ff23[k] =  f23[k]-ctau2*(f23[k]-neq23[k])+(ecprod2[3]-prodscal)*neq23[k];
      ff24[k] =  f24[k]-ctau2*(f24[k]-neq24[k])+(ecprod2[4]-prodscal)*neq24[k];
      ff25[k] =  f25[k]-ctau2*(f25[k]-neq25[k])+(ecprod2[5]-prodscal)*neq25[k];
      ff26[k] =  f26[k]-ctau2*(f26[k]-neq26[k])+(ecprod2[6]-prodscal)*neq26[k];
      ff27[k] =  f27[k]-ctau2*(f27[k]-neq27[k])+(ecprod2[7]-prodscal)*neq27[k];
      ff28[k] =  f28[k]-ctau2*(f28[k]-neq28[k])+(ecprod2[8]-prodscal)*neq28[k];
    }

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:
          f10[k] = ff10[k];
          f11[k] = out1(ff11[nv3[nv3[k]]],ff11[nv3[k]],ff11[k]);
          f12[k] = out1(ff12[nv4[nv4[k]]],ff12[nv4[k]],ff12[k]);
          f13[k] = out1(ff13[nv1[nv1[k]]],ff13[nv1[k]],ff13[k]);
          f14[k] = out1(ff14[nv2[nv2[k]]],ff14[nv2[k]],ff14[k]);
          f15[k] = out1(ff15[nv7[nv7[k]]],ff15[nv7[k]],ff15[k]);
          f16[k] = out1(ff16[nv8[nv8[k]]],ff16[nv8[k]],ff16[k]);
          f17[k] = out1(ff17[nv5[nv5[k]]],ff17[nv5[k]],ff17[k]);
          f18[k] = out1(ff18[nv6[nv6[k]]],ff18[nv6[k]],ff18[k]);

          f20[k] = ff20[k];
          f21[k] = out2(ff21[nv3[nv3[k]]],ff21[nv3[k]],ff21[k]);
          f22[k] = out2(ff22[nv4[nv4[k]]],ff22[nv4[k]],ff22[k]);
          f23[k] = out2(ff23[nv1[nv1[k]]],ff23[nv1[k]],ff23[k]);
          f24[k] = out2(ff24[nv2[nv2[k]]],ff24[nv2[k]],ff24[k]);
          f25[k] = out2(ff25[nv7[nv7[k]]],ff25[nv7[k]],ff25[k]);
          f26[k] = out2(ff26[nv8[nv8[k]]],ff26[nv8[k]],ff26[k]);
          f27[k] = out2(ff27[nv5[nv5[k]]],ff27[nv5[k]],ff27[k]);
          f28[k] = out2(ff28[nv6[nv6[k]]],ff28[nv6[k]],ff28[k]);
          break;

        case 11:
          f10[k] = ff10[k];
          f11[k] = out1(ff11[nv3[nv3[k]]],ff11[nv3[k]],ff11[k]);
          f12[k] = cent1(ff12[nv4[k]],ff12[k],ff12[nv2[k]]);
          f13[k] = out1(ff13[nv1[nv1[k]]],ff13[nv1[k]],ff13[k]);
          f14[k] = out1(ff14[nv2[nv2[k]]],ff14[nv2[k]],ff14[k]);
          f15[k] = cent1(ff15[nv7[k]],ff15[k],ff15[nv5[k]]);
          f16[k] = cent1(ff16[nv8[k]],ff16[k],ff16[nv6[k]]);
          f17[k] = out1(ff17[nv5[nv5[k]]],ff17[nv5[k]],ff17[k]);
          f18[k] = out1(ff18[nv6[nv6[k]]],ff18[nv6[k]],ff18[k]);
```



```c
          f20[k] = ff20[k];
          f21[k] = out2(ff21[nv3[nv3[k]]],ff21[nv3[k]],ff21[k]);
          f22[k] = cent2(ff22[nv4[k]],ff22[k],ff22[nv2[k]]);
          f23[k] = out2(ff23[nv1[nv1[k]]],ff23[nv1[k]],ff23[k]);
          f24[k] = out2(ff24[nv2[nv2[k]]],ff24[nv2[k]],ff24[k]);
          f25[k] = cent2(ff25[nv7[k]],ff25[k],ff25[nv5[k]]);
          f26[k] = cent2(ff26[nv8[k]],ff26[k],ff26[nv6[k]]);
          f27[k] = out2(ff27[nv5[nv5[k]]],ff27[nv5[k]],ff27[k]);
          f28[k] = out2(ff28[nv6[nv6[k]]],ff28[nv6[k]],ff28[k]);
          break;

        case 22:
          f10[k] = ff10[k];
          f11[k] = out1(ff11[nv3[nv3[k]]],ff11[nv3[k]],ff11[k]);
          f12[k] = out1(ff12[nv4[nv4[k]]],ff12[nv4[k]],ff12[k]);
          f13[k] = out1(ff13[nv1[nv1[k]]],ff13[nv1[k]],ff13[k]);
          f14[k] = cent1(ff14[nv2[k]],ff14[k],ff14[nv4[k]]);
          f15[k] = out1(ff15[nv7[nv7[k]]],ff15[nv7[k]],ff15[k]);
          f16[k] = out1(ff16[nv8[nv8[k]]],ff16[nv8[k]],ff16[k]);
          f17[k] = cent1(ff17[nv5[k]],ff17[k],ff17[nv7[k]]);
          f18[k] = cent1(ff18[nv6[k]],ff18[k],ff18[nv8[k]]);

          f20[k] = ff20[k];
          f21[k] = out2(ff21[nv3[nv3[k]]],ff21[nv3[k]],ff21[k]);
          f22[k] = out2(ff22[nv4[nv4[k]]],ff22[nv4[k]],ff22[k]);
          f23[k] = out2(ff23[nv1[nv1[k]]],ff23[nv1[k]],ff23[k]);
          f24[k] = cent2(ff24[nv2[k]],ff24[k],ff24[nv4[k]]);
          f25[k] = out2(ff25[nv7[nv7[k]]],ff25[nv7[k]],ff25[k]);
          f26[k] = out2(ff26[nv8[nv8[k]]],ff26[nv8[k]],ff26[k]);
          f27[k] = cent2(ff27[nv5[k]],ff27[k],ff27[nv7[k]]);
          f28[k] = cent2(ff28[nv6[k]],ff28[k],ff28[nv8[k]]);
          break;

        case 1:
          f10[k] = ff10[k];
          f11[k] = out1(ff11[nv3[nv3[k]]],ff11[nv3[k]],ff11[k]);
          f12[k] = fin1(ff12[k],ff12[nv2[k]],ff12[nv2[nv2[k]]]);
          f13[k] = out1(ff13[nv1[nv1[k]]],ff13[nv1[k]],ff13[k]);
          f14[k] = out1(ff14[nv2[nv2[k]]],ff14[nv2[k]],ff14[k]);
          f15[k] = fin1(ff15[k],ff15[nv5[nv5[k]]],ff15[nv5[k]]);
          f16[k] = fin1(ff16[k],ff16[nv6[k]],ff16[nv6[nv6[k]]]);
          f17[k] = out1(ff17[nv5[nv5[k]]],ff17[nv5[k]],ff17[k]);
          f18[k] = out1(ff18[nv6[nv6[k]]],ff18[nv6[k]],ff18[k]);

          f20[k] = ff20[k];
          f21[k] = out2(ff21[nv3[nv3[k]]],ff21[nv3[k]],ff21[k]);
          f22[k] = fin2(ff22[k],ff22[nv2[k]],ff22[nv2[nv2[k]]]);
          f23[k] = out2(ff23[nv1[nv1[k]]],ff23[nv1[k]],ff23[k]);
          f24[k] = out2(ff24[nv2[nv2[k]]],ff24[nv2[k]],ff24[k]);
          f25[k] = fin2(ff25[k],ff25[nv5[k]],ff25[nv5[nv5[k]]]);
          f26[k] = fin2(ff26[k],ff26[nv6[k]],ff26[nv6[nv6[k]]]);
          f27[k] = out2(ff27[nv5[nv5[k]]],ff27[nv5[k]],ff27[k]);
          f28[k] = out2(ff28[nv6[nv6[k]]],ff28[nv6[k]],ff28[k]);
          break;

        case 2:
          f10[k] = ff10[k];
          f11[k] = out1(ff11[nv3[nv3[k]]],ff11[nv3[k]],ff11[k]);
          f12[k] = out1(ff12[nv4[nv4[k]]],ff12[nv4[k]],ff12[k]);
          f13[k] = out1(ff13[nv1[nv1[k]]],ff13[nv1[k]],ff13[k]);
          f14[k] = fin1(ff14[k],ff14[nv4[k]],ff14[nv4[nv4[k]]]);
          f15[k] = out1(ff15[nv7[nv7[k]]],ff15[nv7[k]],ff15[k]);
          f16[k] = out1(ff16[nv8[nv8[k]]],ff16[nv8[k]],ff16[k]);
          f17[k] = fin1(ff17[k],ff17[nv7[k]],ff17[nv7[nv7[k]]]);
          f18[k] = fin1(ff18[k],ff18[nv8[k]],ff18[nv8[nv8[k]]]);

          f20[k] = ff20[k];
          f21[k] = out2(ff21[nv3[nv3[k]]],ff21[nv3[k]],ff21[k]);
          f22[k] = out2(ff22[nv4[nv4[k]]],ff22[nv4[k]],ff22[k]);
```

```c
          f23[k] = out2(ff23[nv1[nv1[k]]],ff23[nv1[k]],ff23[k]);
          f24[k] = fin2(ff24[k],ff24[nv4[k]],ff24[nv4[nv4[k]]]);
          f25[k] = out2(ff25[nv7[nv7[k]]],ff25[nv7[k]],ff25[k]);
          f26[k] = out2(ff26[nv8[nv8[k]]],ff26[nv8[k]],ff26[k]);
          f27[k] = fin2(ff27[k],ff27[nv7[k]],ff27[nv7[nv7[k]]]);
          f28[k] = fin2(ff28[k],ff28[nv8[k]],ff28[nv8[nv8[k]]]);
          break;

        }
    }
}

void islb_linear(void)
{
  int k;
  double prodscal;
  for(k=0; k<nnodes_all; k++)
    {
      prodscal = csforce_x * uxloc[k] + csforce_y * uxloc[k];

      ff10[k] =  f10[k]-ctau1*(f10[k]-neq10[k])+(ecprod1[0]-prodscal)*neq10[k];
      ff11[k] =  f11[k]-ctau1*(f11[k]-neq11[k])+(ecprod1[1]-prodscal)*neq11[k];
      ff12[k] =  f12[k]-ctau1*(f12[k]-neq12[k])+(ecprod1[2]-prodscal)*neq12[k];
      ff13[k] =  f13[k]-ctau1*(f13[k]-neq13[k])+(ecprod1[3]-prodscal)*neq13[k];
      ff14[k] =  f14[k]-ctau1*(f14[k]-neq14[k])+(ecprod1[4]-prodscal)*neq14[k];
      ff15[k] =  f15[k]-ctau1*(f15[k]-neq15[k])+(ecprod1[5]-prodscal)*neq15[k];
      ff16[k] =  f16[k]-ctau1*(f16[k]-neq16[k])+(ecprod1[6]-prodscal)*neq16[k];
      ff17[k] =  f17[k]-ctau1*(f17[k]-neq17[k])+(ecprod1[7]-prodscal)*neq17[k];
      ff18[k] =  f18[k]-ctau1*(f18[k]-neq18[k])+(ecprod1[8]-prodscal)*neq18[k];

      ff20[k] =  f20[k]-ctau2*(f20[k]-neq20[k])+(ecprod2[0]-prodscal)*neq20[k];
      ff21[k] =  f21[k]-ctau2*(f21[k]-neq21[k])+(ecprod2[1]-prodscal)*neq21[k];
      ff22[k] =  f22[k]-ctau2*(f22[k]-neq22[k])+(ecprod2[2]-prodscal)*neq22[k];
      ff23[k] =  f23[k]-ctau2*(f23[k]-neq23[k])+(ecprod2[3]-prodscal)*neq23[k];
      ff24[k] =  f24[k]-ctau2*(f24[k]-neq24[k])+(ecprod2[4]-prodscal)*neq24[k];
      ff25[k] =  f25[k]-ctau2*(f25[k]-neq25[k])+(ecprod2[5]-prodscal)*neq25[k];
      ff26[k] =  f26[k]-ctau2*(f26[k]-neq26[k])+(ecprod2[6]-prodscal)*neq26[k];

      ff27[k] =  f27[k]-ctau2*(f27[k]-neq27[k])+(ecprod2[7]-prodscal)*neq27[k];
      ff28[k] =  f28[k]-ctau2*(f28[k]-neq28[k])+(ecprod2[8]-prodscal)*neq28[k];
    }

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:

          f10[k] = ff10[k];
          f11[k] = cf11 * ff11[nv3[k]] + oneminus1 * ff11[k];
          f12[k] = cf11 * ff12[nv4[k]] + oneminus1 * ff12[k];
          f13[k] = cf11 * ff13[nv1[k]] + oneminus1 * ff13[k];
          f14[k] = cf11 * ff14[nv2[k]] + oneminus1 * ff14[k];
          f15[k] = cf11 * ff15[nv7[k]] + oneminus1 * ff15[k];
          f16[k] = cf11 * ff16[nv8[k]] + oneminus1 * ff16[k];
          f17[k] = cf11 * ff17[nv5[k]] + oneminus1 * ff17[k];
          f18[k] = cf11 * ff18[nv6[k]] + oneminus1 * ff18[k];

          f20[k] = ff20[k];
          f21[k] = cf12 * ff21[nv3[k]] + oneminus2 * ff21[k];
          f22[k] = cf12 * ff22[nv4[k]] + oneminus2 * ff22[k];
          f23[k] = cf12 * ff23[nv1[k]] + oneminus2 * ff23[k];
          f24[k] = cf12 * ff24[nv2[k]] + oneminus2 * ff24[k];
          f25[k] = cf12 * ff25[nv7[k]] + oneminus2 * ff25[k];
          f26[k] = cf12 * ff26[nv8[k]] + oneminus2 * ff26[k];
          f27[k] = cf12 * ff27[nv5[k]] + oneminus2 * ff27[k];
          f28[k] = cf12 * ff28[nv6[k]] + oneminus2 * ff28[k];

          break;
```



```c
        case 1:

          f10[k] = ff10[k];
          f11[k] = cf11 * ff11[nv3[k]] + oneminus1 * ff11[k];
          f12[k] = oneplus1 * ff12[k] - cf11 * ff12[nv2[k]];
          f13[k] = cf11 * ff13[nv1[k]] + oneminus1 * ff13[k];
          f14[k] = cf11 * ff14[nv2[k]] + oneminus1 * ff14[k];
          f15[k] = oneplus1 * ff15[k] - cf11 * ff15[nv5[k]];
          f16[k] = oneplus1 * ff16[k] - cf11 * ff16[nv6[k]];
          f17[k] = cf11 * ff17[nv5[k]] + oneminus1 * ff17[k];
          f18[k] = cf11 * ff18[nv6[k]] + oneminus1 * ff18[k];

          f20[k] = ff10[k];
          f21[k] = cf12 * ff21[nv3[k]] + oneminus2 * ff21[k];
          f22[k] = oneplus2 * ff22[k] - cf12 * ff22[nv2[k]];
          f23[k] = cf12 * ff23[nv1[k]] + oneminus2 * ff23[k];
          f24[k] = cf12 * ff24[nv2[k]] + oneminus2 * ff24[k];
          f25[k] = oneplus2 * ff25[k] - cf12 * ff25[nv5[k]];
          f26[k] = oneplus2 * ff26[k] - cf12 * ff26[nv6[k]];
          f27[k] = cf12 * ff27[nv5[k]] + oneminus2 * ff27[k];
          f28[k] = cf12 * ff28[nv6[k]] + oneminus2 * ff28[k];

          break;
        case 2:

          f10[k] = ff10[k];
          f11[k] = cf11 * ff11[nv3[k]] + oneminus1 * ff11[k];
          f12[k] = cf11 * ff12[nv4[k]] + oneminus1 * ff12[k];
          f13[k] = cf11 * ff13[nv1[k]] + oneminus1 * ff13[k];
          f14[k] = oneplus1 * ff14[k] - cf11 * ff14[nv4[k]];
          f15[k] = cf11 * ff15[nv7[k]] + oneminus1 * ff15[k];
          f16[k] = cf11 * ff16[nv8[k]] + oneminus1 * ff16[k];
          f17[k] = oneplus1 * ff17[k] - cf11 * ff17[nv7[k]];
          f18[k] = oneplus1 * ff18[k] - cf11 * ff18[nv8[k]];

          f20[k] = ff20[k];
          f21[k] = cf12 * ff21[nv3[k]] + oneminus2 * ff21[k];
          f22[k] = cf12 * ff22[nv4[k]] + oneminus2 * ff22[k];
          f23[k] = cf12 * ff23[nv1[k]] + oneminus2 * ff23[k];
          f24[k] = oneplus2 * ff24[k] - cf12 * ff24[nv4[k]];
          f25[k] = cf12 * ff25[nv7[k]] + oneminus2 * ff25[k];
          f26[k] = cf12 * ff26[nv8[k]] + oneminus2 * ff26[k];
          f27[k] = oneplus2 * ff27[k] - cf12 * ff27[nv7[k]];
          f28[k] = oneplus2 * ff28[k] - cf12 * ff28[nv8[k]];

          break;

        case 3:

          f10[k] = ff10[k];
          f11[k] = oneplus1 * ff11[k] - cf11 * ff11[nv1[k]];
          f12[k] = cf11 * ff12[nv4[k]] + oneminus1 * ff12[k];
          f13[k] = cf11 * ff13[nv1[k]] + oneminus1 * ff13[k];
          f14[k] = cf11 * ff14[nv2[k]] + oneminus1 * ff14[k];
          f15[k] = oneplus1 * ff15[k] - cf11 * ff15[nv5[k]];
          f16[k] = cf11 * ff16[nv8[k]] + oneminus1 * ff16[k];
          f17[k] = cf11 * ff17[nv5[k]] + oneminus1 * ff17[k];
          f18[k] = oneplus1 * ff18[k] - cf11 * ff18[nv8[k]];

          f20[k] = ff20[k];
          f21[k] = oneplus2 * ff21[k] - cf12 * ff21[nv1[k]];
          f22[k] = cf12 * ff22[nv4[k]] + oneminus2 * ff22[k];
          f23[k] = cf12 * ff23[nv1[k]] + oneminus2 * ff23[k];
          f24[k] = cf12 * ff24[nv2[k]] + oneminus2 * ff24[k];
          f25[k] = oneplus2 * ff25[k] - cf12 * ff25[nv5[k]];
          f26[k] = cf12 * ff26[nv8[k]] + oneminus2 * ff26[k];
          f27[k] = cf12 * ff27[nv5[k]] + oneminus2 * ff27[k];
          f28[k] = oneplus2 * ff28[k] - cf12 * ff28[nv8[k]];
```

```
        break;
      case 4:

        f10[k] = ff10[k];
        f11[k] = cfl1 * ff11[nv1[3]] + oneminus1 * ff11[k];
        f12[k] = cfl1 * ff12[nv4[k]] + oneminus1 * ff12[k];
        f13[k] = oneplus1 * ff13[k] - cfl1 * ff13[nv3[k]];
        f14[k] = cfl1 * ff14[nv2[k]] + oneminus1 * ff14[k];
        f15[k] = cfl1 * ff15[nv7[k]] + oneminus1 * ff15[k];
        f16[k] = oneplus1 * ff16[k] - cfl1 * ff16[nv6[k]];
        f17[k] = oneplus1 * ff17[k] - cfl1 * ff17[nv7[k]];
        f18[k] = cfl1 * ff18[nv6[k]] + oneminus1 * ff18[k];

        f20[k] = ff20[k];
        f21[k] = cfl2 * ff21[nv1[3]] + oneminus2 * ff21[k];
        f22[k] = cfl2 * ff22[nv4[k]] + oneminus2 * ff22[k];
        f23[k] = oneplus2 * ff23[k] - cfl2 * ff23[nv3[k]];
        f24[k] = cfl2 * ff24[nv2[k]] + oneminus2 * ff24[k];
        f25[k] = cfl2 * ff25[nv7[k]] + oneminus2 * ff25[k];
        f26[k] = oneplus2 * ff26[k] - cfl2 * ff26[nv6[k]];
        f27[k] = oneplus2 * ff27[k] - cfl2 * ff27[nv7[k]];
        f28[k] = cfl2 * ff28[nv6[k]] + oneminus2 * ff28[k];

        break;
      }
    }
}

void iprop(void)
{
  int k;
  double prodscal;

  for(k=0; k<nnodes_all; k++)
    {
     prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k];

     ff10[k] = f10[k] - ctau1 * ( f10[k] - neq10[k] ) +
       (ecprod1[0] - prodscal) * neq10[k];
     ff11[k] = f11[k] - ctau1 * ( f11[k] - neq11[k] ) +
       (ecprod1[1] - prodscal) * neq11[k];
     ff12[k] = f12[k] - ctau1 * ( f12[k] - neq12[k] ) +
       (ecprod1[2] - prodscal) * neq12[k];
     ff13[k] = f13[k] - ctau1 * ( f13[k] - neq13[k] ) +
       (ecprod1[3] - prodscal) * neq13[k];
     ff14[k] = f14[k] - ctau1 * ( f14[k] - neq14[k] ) +
       (ecprod1[4] - prodscal) * neq14[k];
     ff15[k] = f15[k] - ctau1 * ( f15[k] - neq15[k] ) +
       (ecprod1[5] - prodscal) * neq15[k];
     ff16[k] = f16[k] - ctau1 * ( f16[k] - neq16[k] ) +
       (ecprod1[6] - prodscal) * neq16[k];
     ff17[k] = f17[k] - ctau1 * ( f17[k] - neq17[k] ) +
       (ecprod1[7] - prodscal) * neq17[k];
     ff18[k] = f18[k] - ctau1 * ( f18[k] - neq18[k] ) +
       (ecprod1[8] - prodscal) * neq18[k];

     ff20[k] = f20[k] - ctau2 * ( f20[k] - neq20[k] ) +
       (ecprod2[0] - prodscal) * neq20[k];
     ff21[k] = f21[k] - ctau2 * ( f21[k] - neq21[k] ) +
       (ecprod2[1] - prodscal) * neq21[k];
     ff22[k] = f22[k] - ctau2 * ( f22[k] - neq22[k] ) +
       (ecprod2[2] - prodscal) * neq22[k];
     ff23[k] = f23[k] - ctau2 * ( f23[k] - neq23[k] ) +
       (ecprod2[3] - prodscal) * neq23[k];
     ff24[k] = f24[k] - ctau2 * ( f24[k] - neq24[k] ) +
       (ecprod2[4] - prodscal) * neq24[k];
     ff25[k] = f25[k] - ctau2 * ( f25[k] - neq25[k] ) +
       (ecprod2[5] - prodscal) * neq25[k];
```



```
     ff26[k] = f26[k] - ctau2 * ( f26[k] - neq26[k] ) +
       (ecprod2[6] - prodscal) * neq26[k];
     ff27[k] = f27[k] - ctau2 * ( f27[k] - neq27[k] ) +
       (ecprod2[7] - prodscal) * neq27[k];
     ff28[k] = f28[k] - ctau2 * ( f28[k] - neq28[k] ) +
       (ecprod2[8] - prodscal) * neq28[k];

     /*
       if(k==0)
       printf("iter=%d  k=%d  ff10=%e  ff20=%e\n",iter,k,ff10[k],ff20[k]);
       printf("iter=%d  k=%d  neq10=%e  f10-neq=%e\n",iter,k,neq10[k],
       f10[k]-neq10[k]);
     */

     if(key_init > 2)
       {
         printf("iter=%d  k=%d  gradn1x=%e  gradn1y=%e  gradn2x=%e  gradn2y=%e\n",
                iter,k,gradn1x[k],gradn1y[k],gradn2x[k],gradn2y[k]);
         ff10[k] += kforce * (gradn2x[k] * (ecx1[0] - uxloc[k]) +
                              gradn2y[k] * (ecy1[0] - uyloc[k])) * neq10[k];
         ff11[k] += kforce * (gradn2x[k] * (ecx1[1] - uxloc[k]) +
                              gradn2y[k] * (ecy1[1] - uyloc[k])) * neq11[k];
         ff12[k] += kforce * (gradn2x[k] * (ecx1[2] - uxloc[k]) +
                              gradn2y[k] * (ecy1[2] - uyloc[k])) * neq12[k];
         ff13[k] += kforce * (gradn2x[k] * (ecx1[3] - uxloc[k]) +
                              gradn2y[k] * (ecy1[3] - uyloc[k])) * neq13[k];
         ff14[k] += kforce * (gradn2x[k] * (ecx1[4] - uxloc[k]) +
                              gradn2y[k] * (ecy1[4] - uyloc[k])) * neq14[k];
         ff15[k] += kforce * (gradn2x[k] * (ecx1[5] - uxloc[k]) +
                              gradn2y[k] * (ecy1[5] - uyloc[k])) * neq15[k];
         ff16[k] += kforce * (gradn2x[k] * (ecx1[6] - uxloc[k]) +
                              gradn2y[k] * (ecy1[6] - uyloc[k])) * neq16[k];
         ff17[k] += kforce * (gradn2x[k] * (ecx1[7] - uxloc[k]) +
                              gradn2y[k] * (ecy1[7] - uyloc[k])) * neq17[k];
         ff18[k] += kforce * (gradn2x[k] * (ecx1[8] - uxloc[k]) +
                              gradn2y[k] * (ecy1[8] - uyloc[k])) * neq18[k];

         ff20[k] += kforce * (gradn1x[k] * (ecx2[0] - uxloc[k]) +
                              gradn1y[k] * (ecy2[0] - uyloc[k])) * neq20[k];
         ff21[k] += kforce * (gradn1x[k] * (ecx2[1] - uxloc[k]) +
                              gradn1y[k] * (ecy2[1] - uyloc[k])) * neq21[k];
         ff22[k] += kforce * (gradn1x[k] * (ecx2[2] - uxloc[k]) +
                              gradn1y[k] * (ecy2[2] - uyloc[k])) * neq22[k];
         ff23[k] += kforce * (gradn1x[k] * (ecx2[3] - uxloc[k]) +
                              gradn1y[k] * (ecy2[3] - uyloc[k])) * neq23[k];
         ff24[k] += kforce * (gradn1x[k] * (ecx2[4] - uxloc[k]) +
                              gradn1y[k] * (ecy2[4] - uyloc[k])) * neq24[k];
         ff25[k] += kforce * (gradn1x[k] * (ecx2[5] - uxloc[k]) +
                              gradn1y[k] * (ecy2[5] - uyloc[k])) * neq25[k];
         ff26[k] += kforce * (gradn1x[k] * (ecx2[6] - uxloc[k]) +
                              gradn1y[k] * (ecy2[6] - uyloc[k])) * neq26[k];
         ff27[k] += kforce * (gradn1x[k] * (ecx2[7] - uxloc[k]) +
                              gradn1y[k] * (ecy2[7] - uyloc[k])) * neq27[k];
         ff28[k] += kforce * (gradn1x[k] * (ecx2[8] - uxloc[k]) +
                              gradn1y[k] * (ecy2[8] - uyloc[k])) * neq28[k];
       }

    }
}


void ic(void)
{
  int k;
  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:
```

```
      f10[k] = ff10[k];

      f11[k] = ff11[k] * nc10[1] +
          ff11[nv1[k]] * nc11[1] + ff11[nv2[k]] * nc12[1] +
          ff11[nv3[k]] * nc13[1] + ff11[nv4[k]] * nc14[1] +
          ff11[nv5[k]] * nc15[1] + ff11[nv6[k]] * nc16[1] +
          ff11[nv7[k]] * nc17[1] + ff11[nv8[k]] * nc18[1];

      f12[k] = ff12[k] * nc10[2] +
          ff12[nv1[k]] * nc11[2] + ff12[nv2[k]] * nc12[2] +
          ff12[nv3[k]] * nc13[2] + ff12[nv4[k]] * nc14[2] +
          ff12[nv5[k]] * nc15[2] + ff12[nv6[k]] * nc16[2] +
          ff12[nv7[k]] * nc17[2] + ff12[nv8[k]] * nc18[2];

      f13[k] = ff13[k] * nc10[3] +
          ff13[nv1[k]] * nc11[3] + ff13[nv2[k]] * nc12[3] +
          ff13[nv3[k]] * nc13[3] + ff13[nv4[k]] * nc14[3] +
          ff13[nv5[k]] * nc15[3] + ff13[nv6[k]] * nc16[3] +
          ff13[nv7[k]] * nc17[3] + ff13[nv8[k]] * nc18[3];

      f14[k] = ff14[k] * nc10[4] +
          ff14[nv1[k]] * nc11[4] + ff14[nv2[k]] * nc12[4] +
          ff14[nv3[k]] * nc13[4] + ff14[nv4[k]] * nc14[4] +
          ff14[nv5[k]] * nc15[4] + ff14[nv6[k]] * nc16[4] +
          ff14[nv7[k]] * nc17[4] + ff14[nv8[k]] * nc18[4];

      f15[k] = ff15[k] * nc10[5] +
          ff15[nv1[k]] * nc11[5] + ff15[nv2[k]] * nc12[5] +
          ff15[nv3[k]] * nc13[5] + ff15[nv4[k]] * nc14[5] +
          ff15[nv5[k]] * nc15[5] + ff15[nv6[k]] * nc16[5] +
          ff15[nv7[k]] * nc17[5] + ff15[nv8[k]] * nc18[5];

      f16[k] = ff16[k] * nc10[6] +
          ff16[nv1[k]] * nc11[6] + ff16[nv2[k]] * nc12[6] +
          ff16[nv3[k]] * nc13[6] + ff16[nv4[k]] * nc14[6] +
          ff16[nv5[k]] * nc15[6] + ff16[nv6[k]] * nc16[6] +
          ff16[nv7[k]] * nc17[6] + ff16[nv8[k]] * nc18[6];

      f17[k] = ff17[k] * nc10[7] +
          ff17[nv1[k]] * nc11[7] + ff17[nv2[k]] * nc12[7] +
          ff17[nv3[k]] * nc13[7] + ff17[nv4[k]] * nc14[7] +
          ff17[nv5[k]] * nc15[7] + ff17[nv6[k]] * nc16[7] +
          ff17[nv7[k]] * nc17[7] + ff17[nv8[k]] * nc18[7];

      f18[k] = ff18[k] * nc10[8] +
          ff18[nv1[k]] * nc11[8] + ff18[nv2[k]] * nc12[8] +
          ff18[nv3[k]] * nc13[8] + ff18[nv4[k]] * nc14[8] +
          ff18[nv5[k]] * nc15[8] + ff18[nv6[k]] * nc16[8] +
          ff18[nv7[k]] * nc17[8] + ff18[nv8[k]] * nc18[8];


      f20[k] = ff20[k];

      f21[k] = ff21[k] * nc20[1] +
          ff21[nv1[k]] * nc21[1] + ff21[nv2[k]] * nc22[1] +
          ff21[nv3[k]] * nc23[1] + ff21[nv4[k]] * nc24[1] +
          ff21[nv5[k]] * nc25[1] + ff21[nv6[k]] * nc26[1] +
          ff21[nv7[k]] * nc27[1] + ff21[nv8[k]] * nc28[1];

      f22[k] = ff22[k] * nc20[2] +
          ff22[nv1[k]] * nc21[2] + ff22[nv2[k]] * nc22[2] +
          ff22[nv3[k]] * nc23[2] + ff22[nv4[k]] * nc24[2] +
          ff22[nv5[k]] * nc25[2] + ff22[nv6[k]] * nc26[2] +
          ff22[nv7[k]] * nc27[2] + ff22[nv8[k]] * nc28[2];

      f23[k] = ff23[k] * nc20[3] +
        ff23[nv1[k]] * nc21[3] + ff23[nv2[k]] * nc22[3] +
          ff23[nv3[k]] * nc23[3] + ff23[nv4[k]] * nc24[3] +
          ff23[nv5[k]] * nc25[3] + ff23[nv6[k]] * nc26[3] +
```



```
          ff23[nv7[k]] * nc27[3] + ff23[nv8[k]] * nc28[3];

      f24[k] = ff24[k] * nc20[4] +
          ff24[nv1[k]] * nc21[4] + ff24[nv2[k]] * nc22[4] +
          ff24[nv3[k]] * nc23[4] + ff24[nv4[k]] * nc24[4] +
          ff24[nv5[k]] * nc25[4] + ff24[nv6[k]] * nc26[4] +
          ff24[nv7[k]] * nc27[4] + ff24[nv8[k]] * nc28[4];

      f25[k] = ff25[k] * nc20[5] +
          ff25[nv1[k]] * nc21[5] + ff25[nv2[k]] * nc22[5] +
          ff25[nv3[k]] * nc23[5] + ff25[nv4[k]] * nc24[5] +
          ff25[nv5[k]] * nc25[5] + ff25[nv6[k]] * nc26[5] +
          ff25[nv7[k]] * nc27[5] + ff25[nv8[k]] * nc28[5];

      f26[k] = ff26[k] * nc20[6] +
          ff26[nv1[k]] * nc21[6] + ff26[nv2[k]] * nc22[6] +
          ff26[nv3[k]] * nc23[6] + ff26[nv4[k]] * nc24[6] +
          ff26[nv5[k]] * nc25[6] + ff26[nv6[k]] * nc26[6] +
          ff26[nv7[k]] * nc27[6] + ff26[nv8[k]] * nc28[6];

      f27[k] = ff27[k] * nc20[7] +
          ff27[nv1[k]] * nc21[7] + ff27[nv2[k]] * nc22[7] +
          ff27[nv3[k]] * nc23[7] + ff27[nv4[k]] * nc24[7] +
          ff27[nv5[k]] * nc25[7] + ff27[nv6[k]] * nc26[7] +
          ff27[nv7[k]] * nc27[7] + ff27[nv8[k]] * nc28[7];

      f28[k] = ff28[k] * nc20[8] +
          ff28[nv1[k]] * nc21[8] + ff28[nv2[k]] * nc22[8] +
          ff28[nv3[k]] * nc23[8] + ff28[nv4[k]] * nc24[8] +
          ff28[nv5[k]] * nc25[8] + ff28[nv6[k]] * nc26[8] +
          ff28[nv7[k]] * nc27[8] + ff28[nv8[k]] * nc28[8];

      /*
        if(k==0)
        printf("iter=%d  k=%d  f1=%e  f2=%e\n",iter,k,f10[k],f20[k]);
      */
      break;
    case 1:
      f10[k] = ff10[k];

      f11[k] = ff11[k] * ncbot10[1] + ff11[nv1[k]] * ncbot11[1] +
          ff11[nv2[k]] * ncbot12[1] + ff11[nv3[k]] * ncbot13[1] +
          ff11[nv5[k]] * ncbot15[1] + ff11[nv6[k]] * ncbot16[1];

      f12[k] = ff12[k] * ncbot10[2] + ff12[nv1[k]] * ncbot11[2] +
          ff12[nv2[k]] * ncbot12[2] + ff12[nv3[k]] * ncbot13[2] +
          ff12[nv5[k]] * ncbot15[2] + ff12[nv6[k]] * ncbot16[2];

      f13[k] = ff13[k] * ncbot10[3] + ff13[nv1[k]] * ncbot11[3] +
          ff13[nv2[k]] * ncbot12[3] + ff13[nv3[k]] * ncbot13[3] +
          ff13[nv5[k]] * ncbot15[3] + ff13[nv6[k]] * ncbot16[3];

      f14[k] = ff14[k] * ncbot10[4] + ff14[nv1[k]] * ncbot11[4] +
          ff14[nv2[k]] * ncbot12[4] + ff14[nv3[k]] * ncbot13[4] +
          ff14[nv5[k]] * ncbot15[4] + ff14[nv6[k]] * ncbot16[4];

      f15[k] = ff15[k] * ncbot10[5] + ff15[nv1[k]] * ncbot11[5] +
          ff15[nv2[k]] * ncbot12[5] + ff15[nv3[k]] * ncbot13[5] +
          ff15[nv5[k]] * ncbot15[5] + ff15[nv6[k]] * ncbot16[5];

      f16[k] = ff16[k] * ncbot10[6] + ff16[nv1[k]] * ncbot11[6] +
          ff16[nv2[k]] * ncbot12[6] + ff16[nv3[k]] * ncbot13[6] +
          ff16[nv5[k]] * ncbot15[6] + ff16[nv6[k]] * ncbot16[6];

      f17[k] = ff17[k] * ncbot10[7] + ff17[nv1[k]] * ncbot11[7] +
          ff17[nv2[k]] * ncbot12[7] + ff17[nv3[k]] * ncbot13[7] +
          ff17[nv5[k]] * ncbot15[7] + ff17[nv6[k]] * ncbot16[7];

      f18[k] = ff18[k] * ncbot10[8] + ff18[nv1[k]] * ncbot11[8] +
```

```
        ff18[nv2[k]] * ncbot12[8] + ff18[nv3[k]] * ncbot13[8] +
        ff18[nv5[k]] * ncbot15[8] + ff18[nv6[k]] * ncbot16[8];

    f20[k] = ff20[k];

    f21[k] = ff21[k] * ncbot20[1] + ff21[nv1[k]] * ncbot21[1] +
        ff21[nv2[k]] * ncbot22[1] + ff21[nv3[k]] * ncbot23[1] +
        ff21[nv5[k]] * ncbot25[1] + ff21[nv6[k]] * ncbot26[1];

    f22[k] = ff22[k] * ncbot20[2] + ff22[nv1[k]] * ncbot21[2] +
        ff22[nv2[k]] * ncbot22[2] + ff22[nv3[k]] * ncbot23[2] +
        ff22[nv5[k]] * ncbot25[2] + ff22[nv6[k]] * ncbot26[2];

    f23[k] = ff23[k] * ncbot20[3] + ff23[nv1[k]] * ncbot21[3] +
        ff23[nv2[k]] * ncbot22[3] + ff23[nv3[k]] * ncbot23[3] +
        ff23[nv5[k]] * ncbot25[3] + ff23[nv6[k]] * ncbot26[3];

    f24[k] = ff24[k] * ncbot20[4] + ff24[nv1[k]] * ncbot21[4] +
        ff24[nv2[k]] * ncbot22[4] + ff24[nv3[k]] * ncbot23[4] +
        ff24[nv5[k]] * ncbot25[4] + ff24[nv6[k]] * ncbot26[4];

    f25[k] = ff25[k] * ncbot20[5] + ff25[nv1[k]] * ncbot21[5] +
        ff25[nv2[k]] * ncbot22[5] + ff25[nv3[k]] * ncbot23[5] +
        ff25[nv5[k]] * ncbot25[5] + ff25[nv6[k]] * ncbot26[5];

    f26[k] = ff26[k] * ncbot20[6] + ff26[nv1[k]] * ncbot21[6] +
        ff26[nv2[k]] * ncbot22[6] + ff26[nv3[k]] * ncbot23[6] +
        ff26[nv5[k]] * ncbot25[6] + ff26[nv6[k]] * ncbot26[6];

    f27[k] = ff27[k] * ncbot20[7] + ff27[nv1[k]] * ncbot21[7] +
        ff27[nv2[k]] * ncbot22[7] + ff27[nv3[k]] * ncbot23[7] +
        ff27[nv5[k]] * ncbot25[7] + ff27[nv6[k]] * ncbot26[7];

    f28[k] = ff28[k] * ncbot20[8] + ff28[nv1[k]] * ncbot21[8] +
        ff28[nv2[k]] * ncbot22[8] + ff28[nv3[k]] * ncbot23[8] +
        ff28[nv5[k]] * ncbot25[8] + ff28[nv6[k]] * ncbot26[8];

    break;
  case 2:
    f10[k] = ff10[k];

    f11[k] = ff11[k] * nctop10[1] + ff11[nv1[k]] * nctop11[1] +
        ff11[nv3[k]] * nctop13[1] + ff11[nv4[k]] * nctop14[1] +
        ff11[nv7[k]] * nctop17[1] + ff11[nv8[k]] * nctop18[1];

    f12[k] = ff12[k] * nctop10[2] + ff12[nv1[k]] * nctop11[2] +
        ff12[nv3[k]] * nctop13[2] + ff12[nv4[k]] * nctop14[2] +
        ff12[nv7[k]] * nctop17[2] + ff12[nv8[k]] * nctop18[2];

    f13[k] = ff13[k] * nctop10[3] + ff13[nv1[k]] * nctop11[3] +
        ff13[nv3[k]] * nctop13[3] + ff13[nv4[k]] * nctop14[3] +
        ff13[nv7[k]] * nctop17[3] + ff13[nv8[k]] * nctop18[3];

    f14[k] = ff14[k] * nctop10[4] + ff14[nv1[k]] * nctop11[4] +
        ff14[nv3[k]] * nctop13[4] + ff14[nv4[k]] * nctop14[4] +
        ff14[nv7[k]] * nctop17[4] + ff14[nv8[k]] * nctop18[4];

    f15[k] = ff15[k] * nctop10[5] + ff15[nv1[k]] * nctop11[5] +
        ff15[nv3[k]] * nctop13[5] + ff15[nv4[k]] * nctop14[5] +
        ff15[nv7[k]] * nctop17[5] + ff15[nv8[k]] * nctop18[5];

    f16[k] = ff16[k] * nctop10[6] + ff16[nv1[k]] * nctop11[6] +
        ff16[nv3[k]] * nctop13[6] + ff16[nv4[k]] * nctop14[6] +
        ff16[nv7[k]] * nctop17[6] + ff16[nv8[k]] * nctop18[6];

    f17[k] = ff17[k] * nctop10[7] + ff17[nv1[k]] * nctop11[7] +
        ff17[nv3[k]] * nctop13[7] + ff17[nv4[k]] * nctop14[7] +
        ff17[nv7[k]] * nctop17[7] + ff17[nv8[k]] * nctop18[7];
```



```
    f18[k] = ff18[k] * nctop10[8] + ff18[nv1[k]] * nctop11[8] +
        ff18[nv3[k]] * nctop13[8] + ff18[nv4[k]] * nctop14[8] +
        ff18[nv7[k]] * nctop17[8] + ff18[nv8[k]] * nctop18[8];

    f20[k] = ff20[k];

    f21[k] = ff21[k] * nctop20[1] + ff21[nv1[k]] * nctop21[1] +
        ff21[nv3[k]] * nctop23[1] + ff21[nv4[k]] * nctop24[1] +
        ff21[nv7[k]] * nctop27[1] + ff21[nv8[k]] * nctop28[1];

    f22[k] = ff22[k] * nctop20[2] + ff22[nv1[k]] * nctop21[2] +
        ff22[nv3[k]] * nctop23[2] + ff22[nv4[k]] * nctop24[2] +
        ff22[nv7[k]] * nctop27[2] + ff22[nv8[k]] * nctop28[2];

    f23[k] = ff23[k] * nctop20[3] + ff23[nv1[k]] * nctop21[3] +
        ff23[nv3[k]] * nctop23[3] + ff23[nv4[k]] * nctop24[3] +
        ff23[nv7[k]] * nctop27[3] + ff23[nv8[k]] * nctop28[3];

    f24[k] = ff24[k] * nctop20[4] + ff24[nv1[k]] * nctop21[4] +
        ff24[nv3[k]] * nctop23[4] + ff24[nv4[k]] * nctop24[4] +
        ff24[nv7[k]] * nctop27[4] + ff24[nv8[k]] * nctop28[4];

    f25[k] = ff25[k] * nctop20[5] + ff25[nv1[k]] * nctop21[5] +
        ff25[nv3[k]] * nctop23[5] + ff25[nv4[k]] * nctop24[5] +
        ff25[nv7[k]] * nctop27[5] + ff25[nv8[k]] * nctop28[5];

    f26[k] = ff26[k] * nctop20[6] + ff26[nv1[k]] * nctop21[6] +
        ff26[nv3[k]] * nctop23[6] + ff26[nv4[k]] * nctop24[6] +
        ff26[nv7[k]] * nctop27[6] + ff26[nv8[k]] * nctop28[6];

    f27[k] = ff27[k] * nctop20[7] + ff27[nv1[k]] * nctop21[7] +
        ff27[nv3[k]] * nctop23[7] + ff27[nv4[k]] * nctop24[7] +
        ff27[nv7[k]] * nctop27[7] + ff27[nv8[k]] * nctop28[7];

    f28[k] = ff28[k] * nctop20[8] + ff28[nv1[k]] * nctop21[8] +
        ff28[nv3[k]] * nctop23[8] + ff28[nv4[k]] * nctop24[8] +
        ff28[nv7[k]] * nctop27[8] + ff28[nv8[k]] * nctop28[8];

    break;
  case 3:
    f10[k] = ff10[k];

    f11[k] = ff11[k] * ncleft10[1] + ff11[nv1[k]] * ncleft11[1] +
        ff11[nv5[k]] * ncleft15[1] + ff11[nv2[k]] * ncleft12[1] +
        ff11[nv4[k]] * ncleft14[1] + ff11[nv8[k]] * ncleft18[1];

    f12[k] = ff12[k] * ncleft10[2] + ff12[nv1[k]] * ncleft11[2] +
        ff12[nv5[k]] * ncleft15[2] + ff12[nv2[k]] * ncleft12[2] +
        ff12[nv4[k]] * ncleft14[2] + ff12[nv8[k]] * ncleft18[2];

    f13[k] = ff13[k] * ncleft10[3] + ff13[nv1[k]] * ncleft11[3] +
        ff13[nv5[k]] * ncleft15[3] + ff13[nv2[k]] * ncleft12[3] +
        ff13[nv4[k]] * ncleft14[3] + ff13[nv8[k]] * ncleft18[3];

    f14[k] = ff14[k] * ncleft10[4] + ff14[nv1[k]] * ncleft11[4] +
        ff14[nv5[k]] * ncleft15[4] + ff14[nv2[k]] * ncleft12[4] +
        ff14[nv4[k]] * ncleft14[4] + ff14[nv8[k]] * ncleft18[4];

    f15[k] = ff15[k] * ncleft10[5] + ff15[nv1[k]] * ncleft11[5] +
        ff15[nv5[k]] * ncleft15[5] + ff15[nv2[k]] * ncleft12[5] +
        ff15[nv4[k]] * ncleft14[5] + ff15[nv8[k]] * ncleft18[5];

    f16[k] = ff16[k] * ncleft10[6] + ff16[nv1[k]] * ncleft11[6] +
        ff16[nv5[k]] * ncleft15[6] + ff16[nv2[k]] * ncleft12[6] +
        ff16[nv4[k]] * ncleft14[6] + ff16[nv8[k]] * ncleft18[6];

    f17[k] = ff17[k] * ncleft10[7] + ff17[nv1[k]] * ncleft11[7] +
        ff17[nv5[k]] * ncleft15[7] + ff17[nv2[k]] * ncleft12[7] +
        ff17[nv4[k]] * ncleft14[7] + ff17[nv8[k]] * ncleft18[7];
```

```
    f18[k] = ff18[k] * ncleft10[8] + ff18[nv1[k]] * ncleft11[8] +
        ff18[nv5[k]] * ncleft15[8] + ff18[nv2[k]] * ncleft12[8] +
        ff18[nv4[k]] * ncleft14[8] + ff18[nv8[k]] * ncleft18[8];

    f20[k] = ff20[k];

    f21[k] = ff21[k] * ncleft20[1] + ff21[nv1[k]] * ncleft21[1] +
        ff21[nv5[k]] * ncleft25[1] + ff21[nv2[k]] * ncleft22[1] +
        ff21[nv4[k]] * ncleft24[1] + ff21[nv8[k]] * ncleft28[1];

    f22[k] = ff22[k] * ncleft20[2] + ff22[nv1[k]] * ncleft21[2] +
        ff22[nv5[k]] * ncleft25[2] + ff22[nv2[k]] * ncleft22[2] +
        ff22[nv4[k]] * ncleft24[2] + ff22[nv8[k]] * ncleft28[2];

    f23[k] = ff23[k] * ncleft20[3] + ff23[nv1[k]] * ncleft21[3] +
        ff23[nv5[k]] * ncleft25[3] + ff23[nv2[k]] * ncleft22[3] +
        ff23[nv4[k]] * ncleft24[3] + ff23[nv8[k]] * ncleft28[3];

    f24[k] = ff24[k] * ncleft20[4] + ff24[nv1[k]] * ncleft21[4] +
        ff24[nv5[k]] * ncleft25[4] + ff24[nv2[k]] * ncleft22[4] +
        ff24[nv4[k]] * ncleft24[4] + ff24[nv8[k]] * ncleft28[4];

    f25[k] = ff25[k] * ncleft20[5] + ff25[nv1[k]] * ncleft21[5] +
        ff25[nv5[k]] * ncleft25[5] + ff25[nv2[k]] * ncleft22[5] +
        ff25[nv4[k]] * ncleft24[5] + ff25[nv8[k]] * ncleft28[5];

    f26[k] = ff26[k] * ncleft20[6] + ff26[nv1[k]] * ncleft21[6] +
        ff26[nv5[k]] * ncleft25[6] + ff26[nv2[k]] * ncleft22[6] +
        ff26[nv4[k]] * ncleft24[6] + ff26[nv8[k]] * ncleft28[6];

    f27[k] = ff27[k] * ncleft20[7] + ff27[nv1[k]] * ncleft21[7] +
        ff27[nv5[k]] * ncleft25[7] + ff27[nv2[k]] * ncleft22[7] +
        ff27[nv4[k]] * ncleft24[7] + ff27[nv8[k]] * ncleft28[7];

    f28[k] = ff28[k] * ncleft20[8] + ff28[nv1[k]] * ncleft21[8] +
        ff28[nv5[k]] * ncleft25[8] + ff28[nv2[k]] * ncleft22[8] +
        ff28[nv4[k]] * ncleft24[8] + ff28[nv8[k]] * ncleft28[8];

    break;
  case 4:
    f10[k] = ff10[k];

    f11[k] = ff11[k] * ncright10[1] + ff11[nv3[k]] * ncright13[1] +
        ff11[nv6[k]] * ncright16[1] + ff11[nv2[k]] * ncright12[1] +
        ff11[nv4[k]] * ncright14[1] + ff11[nv7[k]] * ncright17[1];

    f12[k] = ff12[k] * ncright10[2] + ff12[nv3[k]] * ncright13[2] +
        ff12[nv6[k]] * ncright16[2] + ff12[nv2[k]] * ncright12[2] +
        ff12[nv4[k]] * ncright14[2] + ff12[nv7[k]] * ncright17[2];

    f13[k] = ff13[k] * ncright10[3] + ff13[nv3[k]] * ncright13[3] +
        ff13[nv6[k]] * ncright16[3] + ff13[nv2[k]] * ncright12[3] +
        ff13[nv4[k]] * ncright14[3] + ff13[nv7[k]] * ncright17[3];

    f14[k] = ff14[k] * ncright10[4] + ff14[nv3[k]] * ncright13[4] +
        ff14[nv6[k]] * ncright16[4] + ff14[nv2[k]] * ncright12[4] +
        ff14[nv4[k]] * ncright14[4] + ff14[nv7[k]] * ncright17[4];

    f15[k] = ff15[k] * ncright10[5] + ff15[nv3[k]] * ncright13[5] +
        ff15[nv6[k]] * ncright16[5] + ff15[nv2[k]] * ncright12[5] +
        ff15[nv4[k]] * ncright14[5] + ff15[nv7[k]] * ncright17[5];

    f16[k] = ff16[k] * ncright10[6] + ff16[nv3[k]] * ncright13[6] +
        ff16[nv6[k]] * ncright16[6] + ff16[nv2[k]] * ncright12[6] +
        ff16[nv4[k]] * ncright14[6] + ff16[nv7[k]] * ncright17[6];

    f17[k] = ff17[k] * ncright10[7] + ff17[nv3[k]] * ncright13[7] +
        ff17[nv6[k]] * ncright16[7] + ff17[nv2[k]] * ncright12[7] +
```



```
        ff17[nv4[k]] * ncright14[7] + ff17[nv7[k]] * ncright17[7];

    f18[k] = ff18[k] * ncright10[8] + ff18[nv3[k]] * ncright13[8] +
        ff18[nv6[k]] * ncright16[8] + ff18[nv2[k]] * ncright12[8] +
        ff18[nv4[k]] * ncright14[8] + ff18[nv7[k]] * ncright17[8];

    f20[k] = ff20[k];

    f21[k] = ff21[k] * ncright20[1] + ff21[nv3[k]] * ncright23[1] +
        ff21[nv6[k]] * ncright26[1] + ff21[nv2[k]] * ncright22[1] +
        ff21[nv4[k]] * ncright24[1] + ff21[nv7[k]] * ncright27[1];

    f22[k] = ff22[k] * ncright20[2] + ff22[nv3[k]] * ncright23[2] +
        ff22[nv6[k]] * ncright26[2] + ff22[nv2[k]] * ncright22[2] +
        ff22[nv4[k]] * ncright24[2] + ff22[nv7[k]] * ncright27[2];

    f23[k] = ff23[k] * ncright20[3] + ff23[nv3[k]] * ncright23[3] +
        ff23[nv6[k]] * ncright26[3] + ff23[nv2[k]] * ncright22[3] +
        ff23[nv4[k]] * ncright24[3] + ff23[nv7[k]] * ncright27[3];

    f24[k] = ff24[k] * ncright20[4] + ff24[nv3[k]] * ncright23[4] +
        ff24[nv6[k]] * ncright26[4] + ff24[nv2[k]] * ncright22[4] +
        ff24[nv4[k]] * ncright24[4] + ff24[nv7[k]] * ncright27[4];

    f25[k] = ff25[k] * ncright20[5] + ff25[nv3[k]] * ncright23[5] +
        ff25[nv6[k]] * ncright26[5] + ff25[nv2[k]] * ncright22[5] +
        ff25[nv4[k]] * ncright24[5] + ff25[nv7[k]] * ncright27[5];

    f26[k] = ff26[k] * ncright20[6] + ff26[nv3[k]] * ncright23[6] +
        ff26[nv6[k]] * ncright26[6] + ff26[nv2[k]] * ncright22[6] +
        ff26[nv4[k]] * ncright24[6] + ff26[nv7[k]] * ncright27[6];

    f27[k] = ff27[k] * ncright20[7] + ff27[nv3[k]] * ncright23[7] +
        ff27[nv6[k]] * ncright26[7] + ff27[nv2[k]] * ncright22[7] +
        ff27[nv4[k]] * ncright24[7] + ff27[nv7[k]] * ncright27[7];

    f28[k] = ff28[k] * ncright20[8] + ff28[nv3[k]] * ncright23[8] +
        ff28[nv6[k]] * ncright26[8] + ff28[nv2[k]] * ncright22[8] +
        ff28[nv4[k]] * ncright24[8] + ff28[nv7[k]] * ncright27[8];

    break;
   }
  }
}

void iup(void)
{
  int k;
  for(k=0; k<nnodes_all; k++)
    {
     f10[k] = ff10[k];

     f11[k] = ff11[k] * nu15[1] +
       ff11[nv3[k]] * nu12[1] + ff11[nv3[nv3[k]]] * nu16[1] +
       ff11[nv4[k]] * nu11[1] + ff11[nv7[k]] * nu10[1] +
       ff11[nv3[nv7[k]]] * nu13[1] + ff11[nv4[nv4[k]]] * nu18[1] +
       ff11[nv3[nv4[nv4[k]]]] * nu14[1] + ff11[nv7[nv7[k]]] * nu17[1];

     f12[k] = ff12[k] * nu15[2] +
       ff12[nv3[k]] * nu12[2] + ff12[nv3[nv3[k]]] * nu16[2] +
       ff12[nv4[k]] * nu11[2] + ff12[nv7[k]] * nu10[2] +
       ff12[nv3[nv7[k]]] * nu13[2] + ff12[nv4[nv4[k]]] * nu18[2] +
       ff12[nv3[nv4[nv4[k]]]] * nu14[2] + ff12[nv7[nv7[k]]] * nu17[2];

     f13[k] = ff13[k] * nu15[3] +
       ff13[nv3[k]] * nu12[3] + ff13[nv3[nv3[k]]] * nu16[3] +
       ff13[nv4[k]] * nu11[3] + ff13[nv7[k]] * nu10[3] +
       ff13[nv3[nv7[k]]] * nu13[3] + ff13[nv4[nv4[k]]] * nu18[3] +
       ff13[nv3[nv4[nv4[k]]]] * nu14[3] + ff13[nv7[nv7[k]]] * nu17[3];
```

```
f14[k] = ff14[k] * nu15[4] +
  ff14[nv3[k]] * nu12[4] + ff14[nv3[nv3[k]]] * nu16[4] +
  ff14[nv4[k]] * nu11[4] + ff14[nv7[k]] * nu10[4] +
  ff14[nv3[nv7[k]]] * nu13[4] + ff14[nv4[nv4[k]]] * nu18[4] +
  ff14[nv3[nv4[nv4[k]]]] * nu14[4] + ff14[nv7[nv7[k]]] * nu17[4];

f15[k] = ff15[k] * nu15[5] +
  ff15[nv3[k]] * nu12[5] + ff15[nv3[nv3[k]]] * nu16[5] +
  ff15[nv4[k]] * nu11[5] + ff15[nv7[k]] * nu10[5] +
  ff15[nv3[nv7[k]]] * nu13[5] + ff15[nv4[nv4[k]]] * nu18[5] +
  ff15[nv3[nv4[nv4[k]]]] * nu14[5] + ff15[nv7[nv7[k]]] * nu17[5];

f16[k] = ff16[k] * nu15[6] +
  ff16[nv3[k]] * nu12[6] + ff16[nv3[nv3[k]]] * nu16[6] +
  ff16[nv4[k]] * nu11[6] + ff16[nv7[k]] * nu10[6] +
  ff16[nv3[nv7[k]]] * nu13[6] + ff16[nv4[nv4[k]]] * nu18[6] +
  ff16[nv3[nv4[nv4[k]]]] * nu14[6] + ff16[nv7[nv7[k]]] * nu17[6];

f17[k] = ff17[k] * nu15[7] +
  ff17[nv3[k]] * nu12[7] + ff17[nv3[nv3[k]]] * nu16[7] +
  ff17[nv4[k]] * nu11[7] + ff17[nv7[k]] * nu10[7] +
  ff17[nv3[nv7[k]]] * nu13[7] + ff17[nv4[nv4[k]]] * nu18[7] +
  ff17[nv3[nv4[nv4[k]]]] * nu14[7] + ff17[nv7[nv7[k]]] * nu17[7];

f18[k] = ff18[k] * nu15[8] +
  ff18[nv3[k]] * nu12[8] + ff18[nv3[nv3[k]]] * nu16[8] +
  ff18[nv4[k]] * nu11[8] + ff18[nv7[k]] * nu10[8] +
  ff18[nv3[nv7[k]]] * nu13[8] + ff18[nv4[nv4[k]]] * nu18[8] +
  ff18[nv3[nv4[nv4[k]]]] * nu14[8] + ff18[nv7[nv7[k]]] * nu17[8];

f20[k] = ff20[k];

f21[k] = ff21[k] * nu25[1] +
  ff21[nv3[k]] * nu22[1] + ff21[nv3[nv3[k]]] * nu26[1] +
  ff21[nv4[k]] * nu21[1] + ff21[nv7[k]] * nu20[1] +
  ff21[nv3[nv7[k]]] * nu23[1] + ff21[nv4[nv4[k]]] * nu28[1] +
  ff21[nv3[nv4[nv4[k]]]] * nu24[1] + ff21[nv7[nv7[k]]] * nu27[1];

f22[k] = ff22[k] * nu25[2] +
  ff22[nv3[k]] * nu22[2] + ff22[nv3[nv3[k]]] * nu26[2] +
  ff22[nv4[k]] * nu21[2] + ff22[nv7[k]] * nu20[2] +
  ff22[nv3[nv7[k]]] * nu23[2] + ff22[nv4[nv4[k]]] * nu28[2] +
  ff22[nv3[nv4[nv4[k]]]] * nu24[2] + ff22[nv7[nv7[k]]] * nu27[2];

f23[k] = ff23[k] * nu25[3] +
  ff23[nv3[k]] * nu22[3] + ff23[nv3[nv3[k]]] * nu26[3] +
  ff23[nv4[k]] * nu21[3] + ff23[nv7[k]] * nu20[3] +
  ff23[nv3[nv7[k]]] * nu23[3] + ff23[nv4[nv4[k]]] * nu28[3] +
  ff23[nv3[nv4[nv4[k]]]] * nu24[3] + ff23[nv7[nv7[k]]] * nu27[3];

f24[k] = ff24[k] * nu25[4] +
  ff24[nv3[k]] * nu22[4] + ff24[nv3[nv3[k]]] * nu26[4] +
  ff24[nv4[k]] * nu21[4] + ff24[nv7[k]] * nu20[4] +
  ff24[nv3[nv7[k]]] * nu23[4] + ff24[nv4[nv4[k]]] * nu28[4] +
  ff24[nv3[nv4[nv4[k]]]] * nu24[4] + ff24[nv7[nv7[k]]] * nu27[4];

f25[k] = ff25[k] * nu25[5] +
  ff25[nv3[k]] * nu22[5] + ff25[nv3[nv3[k]]] * nu26[5] +
  ff25[nv4[k]] * nu21[5] + ff25[nv7[k]] * nu20[5] +
  ff25[nv3[nv7[k]]] * nu23[5] + ff25[nv4[nv4[k]]] * nu28[5] +
  ff25[nv3[nv4[nv4[k]]]] * nu24[5] + ff25[nv7[nv7[k]]] * nu27[5];

f26[k] = ff26[k] * nu25[6] +
  ff26[nv3[k]] * nu22[6] + ff26[nv3[nv3[k]]] * nu26[6] +
  ff26[nv4[k]] * nu21[6] + ff26[nv7[k]] * nu20[6] +
  ff26[nv3[nv7[k]]] * nu23[6] + ff26[nv4[nv4[k]]] * nu28[6] +
  ff26[nv3[nv4[nv4[k]]]] * nu24[6] + ff26[nv7[nv7[k]]] * nu27[6];
```



```
      f27[k] = ff27[k] * nu25[7] +
        ff27[nv3[k]] * nu22[7] + ff27[nv3[nv3[k]]] * nu26[7] +
        ff27[nv4[k]] * nu21[7] + ff27[nv7[k]] * nu20[7] +
        ff27[nv3[nv7[k]]] * nu23[7] + ff27[nv4[nv4[k]]] * nu28[7] +
        ff27[nv3[nv4[nv4[k]]]] * nu24[7] + ff27[nv7[nv7[k]]] * nu27[7];

      f28[k] = ff28[k] * nu25[8] +
        ff28[nv3[k]] * nu22[8] + ff28[nv3[nv3[k]]] * nu26[8] +
        ff28[nv4[k]] * nu21[8] + ff28[nv7[k]] * nu20[8] +
        ff28[nv3[nv7[k]]] * nu23[8] + ff28[nv4[nv4[k]]] * nu28[8] +
        ff28[nv3[nv4[nv4[k]]]] * nu24[8] + ff28[nv7[nv7[k]]] * nu27[8];

      /*
      if(k==0)
        printf("iter=%d  k=%d  f1=%e  f2=%e\n",iter,k,f10[k],f20[k]);
      */
    }
}

void ifd(void)
{
  int k;
  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:
          ff10[k] = f10[k];

          ff11[k] = f11[k] * nc10[1] +
               f11[nv1[k]] * nc11[1] + f11[nv2[k]] * nc12[1] +
               f11[nv3[k]] * nc13[1] + f11[nv4[k]] * nc14[1] +
               f11[nv5[k]] * nc15[1] + f11[nv6[k]] * nc16[1] +
               f11[nv7[k]] * nc17[1] + f11[nv8[k]] * nc18[1];

          ff12[k] = f12[k] * nc10[2] +
               f12[nv1[k]] * nc11[2] + f12[nv2[k]] * nc12[2] +
               f12[nv3[k]] * nc13[2] + f12[nv4[k]] * nc14[2] +
               f12[nv5[k]] * nc15[2] + f12[nv6[k]] * nc16[2] +
               f12[nv7[k]] * nc17[2] + f12[nv8[k]] * nc18[2];

          ff13[k] = f13[k] * nc10[3] +
               f13[nv1[k]] * nc11[3] + f13[nv2[k]] * nc12[3] +
               f13[nv3[k]] * nc13[3] + f13[nv4[k]] * nc14[3] +
               f13[nv5[k]] * nc15[3] + f13[nv6[k]] * nc16[3] +
               f13[nv7[k]] * nc17[3] + f13[nv8[k]] * nc18[3];

          ff14[k] = f14[k] * nc10[4] +
               f14[nv1[k]] * nc11[4] + f14[nv2[k]] * nc12[4] +
               f14[nv3[k]] * nc13[4] + f14[nv4[k]] * nc14[4] +
               f14[nv5[k]] * nc15[4] + f14[nv6[k]] * nc16[4] +
               f14[nv7[k]] * nc17[4] + f14[nv8[k]] * nc18[4];

          ff15[k] = f15[k] * nc10[5] +
               f15[nv1[k]] * nc11[5] + f15[nv2[k]] * nc12[5] +
               f15[nv3[k]] * nc13[5] + f15[nv4[k]] * nc14[5] +
               f15[nv5[k]] * nc15[5] + f15[nv6[k]] * nc16[5] +
               f15[nv7[k]] * nc17[5] + f15[nv8[k]] * nc18[5];

          ff16[k] = f16[k] * nc10[6] +
               f16[nv1[k]] * nc11[6] + f16[nv2[k]] * nc12[6] +
               f16[nv3[k]] * nc13[6] + f16[nv4[k]] * nc14[6] +
               f16[nv5[k]] * nc15[6] + f16[nv6[k]] * nc16[6] +
               f16[nv7[k]] * nc17[6] + f16[nv8[k]] * nc18[6];

          ff17[k] = f17[k] * nc10[7] +
               f17[nv1[k]] * nc11[7] + f17[nv2[k]] * nc12[7] +
               f17[nv3[k]] * nc13[7] + f17[nv4[k]] * nc14[7] +
               f17[nv5[k]] * nc15[7] + f17[nv6[k]] * nc16[7] +
```

```
        f17[nv7[k]] * nc17[7] + f17[nv8[k]] * nc18[7];

    ff18[k] = f18[k] * nc10[8] +
        f18[nv1[k]] * nc11[8] + f18[nv2[k]] * nc12[8] +
        f18[nv3[k]] * nc13[8] + f18[nv4[k]] * nc14[8] +
        f18[nv5[k]] * nc15[8] + f18[nv6[k]] * nc16[8] +
        f18[nv7[k]] * nc17[8] + f18[nv8[k]] * nc18[8];


    ff20[k] = f20[k];

    ff21[k] = f21[k] * nc20[1] +
        f21[nv1[k]] * nc21[1] + f21[nv2[k]] * nc22[1] +
        f21[nv3[k]] * nc23[1] + f21[nv4[k]] * nc24[1] +
        f21[nv5[k]] * nc25[1] + f21[nv6[k]] * nc26[1] +
        f21[nv7[k]] * nc27[1] + f21[nv8[k]] * nc28[1];

    ff22[k] = f22[k] * nc20[2] +
        f22[nv1[k]] * nc21[2] + f22[nv2[k]] * nc22[2] +
        f22[nv3[k]] * nc23[2] + f22[nv4[k]] * nc24[2] +
        f22[nv5[k]] * nc25[2] + f22[nv6[k]] * nc26[2] +
        f22[nv7[k]] * nc27[2] + f22[nv8[k]] * nc28[2];

    ff23[k] = f23[k] * nc20[3] +
        f23[nv1[k]] * nc21[3] + f23[nv2[k]] * nc22[3] +
        f23[nv3[k]] * nc23[3] + f23[nv4[k]] * nc24[3] +
        f23[nv5[k]] * nc25[3] + f23[nv6[k]] * nc26[3] +
        f23[nv7[k]] * nc27[3] + f23[nv8[k]] * nc28[3];

    ff24[k] = f24[k] * nc20[4] +
        f24[nv1[k]] * nc21[4] + f24[nv2[k]] * nc22[4] +
        f24[nv3[k]] * nc23[4] + f24[nv4[k]] * nc24[4] +
        f24[nv5[k]] * nc25[4] + f24[nv6[k]] * nc26[4] +
        f24[nv7[k]] * nc27[4] + f24[nv8[k]] * nc28[4];

    ff25[k] = f25[k] * nc20[5] +
        f25[nv1[k]] * nc21[5] + f25[nv2[k]] * nc22[5] +
        f25[nv3[k]] * nc23[5] + f25[nv4[k]] * nc24[5] +
        f25[nv5[k]] * nc25[5] + f25[nv6[k]] * nc26[5] +
        f25[nv7[k]] * nc27[5] + f25[nv8[k]] * nc28[5];

    ff26[k] = f26[k] * nc20[6] +
        f26[nv1[k]] * nc21[6] + f26[nv2[k]] * nc22[6] +
        f26[nv3[k]] * nc23[6] + f26[nv4[k]] * nc24[6] +
        f26[nv5[k]] * nc25[6] + f26[nv6[k]] * nc26[6] +
        f26[nv7[k]] * nc27[6] + f26[nv8[k]] * nc28[6];

    ff27[k] = f27[k] * nc20[7] +
        f27[nv1[k]] * nc21[7] + f27[nv2[k]] * nc22[7] +
        f27[nv3[k]] * nc23[7] + f27[nv4[k]] * nc24[7] +
        f27[nv5[k]] * nc25[7] + f27[nv6[k]] * nc26[7] +
        f27[nv7[k]] * nc27[7] + f27[nv8[k]] * nc28[7];

    ff28[k] = f28[k] * nc20[8] +
        f28[nv1[k]] * nc21[8] + f28[nv2[k]] * nc22[8] +
        f28[nv3[k]] * nc23[8] + f28[nv4[k]] * nc24[8] +
        f28[nv5[k]] * nc25[8] + f28[nv6[k]] * nc26[8] +
        f28[nv7[k]] * nc27[8] + f28[nv8[k]] * nc28[8];
break;
  case 1:
    ff10[k] = f10[k];

    ff11[k] = f11[k] * ncbot10[1] + f11[nv1[k]] * ncbot11[1] +
        f11[nv2[k]] * ncbot12[1] + f11[nv3[k]] * ncbot13[1] +
        f11[nv5[k]] * ncbot15[1] + f11[nv6[k]] * ncbot16[1];

    ff12[k] = f12[k] * ncbot10[2] + f12[nv1[k]] * ncbot11[2] +
        f12[nv2[k]] * ncbot12[2] + f12[nv3[k]] * ncbot13[2] +
        f12[nv5[k]] * ncbot15[2] + f12[nv6[k]] * ncbot16[2];
```



```
    ff13[k] = f13[k] * ncbot10[3] + f13[nv1[k]] * ncbot11[3] +
        f13[nv2[k]] * ncbot12[3] + f13[nv3[k]] * ncbot13[3] +
        f13[nv5[k]] * ncbot15[3] + f13[nv6[k]] * ncbot16[3];

    ff14[k] = f14[k] * ncbot10[4] + f14[nv1[k]] * ncbot11[4] +
        f14[nv2[k]] * ncbot12[4] + f14[nv3[k]] * ncbot13[4] +
        f14[nv5[k]] * ncbot15[4] + f14[nv6[k]] * ncbot16[4];

    ff15[k] = f15[k] * ncbot10[5] + f15[nv1[k]] * ncbot11[5] +
        f15[nv2[k]] * ncbot12[5] + f15[nv3[k]] * ncbot13[5] +
        f15[nv5[k]] * ncbot15[5] + f15[nv6[k]] * ncbot16[5];

    ff16[k] = f16[k] * ncbot10[6] + f16[nv1[k]] * ncbot11[6] +
        f16[nv2[k]] * ncbot12[6] + f16[nv3[k]] * ncbot13[6] +
        f16[nv5[k]] * ncbot15[6] + f16[nv6[k]] * ncbot16[6];

    ff17[k] = f17[k] * ncbot10[7] + f17[nv1[k]] * ncbot11[7] +
        f17[nv2[k]] * ncbot12[7] + f17[nv3[k]] * ncbot13[7] +
        f17[nv5[k]] * ncbot15[7] + f17[nv6[k]] * ncbot16[7];

    ff18[k] = f18[k] * ncbot10[8] + f18[nv1[k]] * ncbot11[8] +
        f18[nv2[k]] * ncbot12[8] + f18[nv3[k]] * ncbot13[8] +
        f18[nv5[k]] * ncbot15[8] + f18[nv6[k]] * ncbot16[8];

    ff20[k] = f20[k];

    ff21[k] = f21[k] * ncbot20[1] + f21[nv1[k]] * ncbot21[1] +
        f21[nv2[k]] * ncbot22[1] + f21[nv3[k]] * ncbot23[1] +
        f21[nv5[k]] * ncbot25[1] + f21[nv6[k]] * ncbot26[1];

    ff22[k] = f22[k] * ncbot20[2] + f22[nv1[k]] * ncbot21[2] +
        f22[nv2[k]] * ncbot22[2] + f22[nv3[k]] * ncbot23[2] +
        f22[nv5[k]] * ncbot25[2] + f22[nv6[k]] * ncbot26[2];

    ff23[k] = f23[k] * ncbot20[3] + f23[nv1[k]] * ncbot21[3] +
        f23[nv2[k]] * ncbot22[3] + f23[nv3[k]] * ncbot23[3] +
        f23[nv5[k]] * ncbot25[3] + f23[nv6[k]] * ncbot26[3];

    ff24[k] = f24[k] * ncbot20[4] + f24[nv1[k]] * ncbot21[4] +
        f24[nv2[k]] * ncbot22[4] + f24[nv3[k]] * ncbot23[4] +
        f24[nv5[k]] * ncbot25[4] + f24[nv6[k]] * ncbot26[4];

    ff25[k] = f25[k] * ncbot20[5] + f25[nv1[k]] * ncbot21[5] +
        f25[nv2[k]] * ncbot22[5] + f25[nv3[k]] * ncbot23[5] +
        f25[nv5[k]] * ncbot25[5] + f25[nv6[k]] * ncbot26[5];

    ff26[k] = f26[k] * ncbot20[6] + f26[nv1[k]] * ncbot21[6] +
        f26[nv2[k]] * ncbot22[6] + f26[nv3[k]] * ncbot23[6] +
        f26[nv5[k]] * ncbot25[6] + f26[nv6[k]] * ncbot26[6];

    ff27[k] = f27[k] * ncbot20[7] + f27[nv1[k]] * ncbot21[7] +
        f27[nv2[k]] * ncbot22[7] + f27[nv3[k]] * ncbot23[7] +
        f27[nv5[k]] * ncbot25[7] + f27[nv6[k]] * ncbot26[7];

    ff28[k] = f28[k] * ncbot20[8] + f28[nv1[k]] * ncbot21[8] +
        f28[nv2[k]] * ncbot22[8] + f28[nv3[k]] * ncbot23[8] +
        f28[nv5[k]] * ncbot25[8] + f28[nv6[k]] * ncbot26[8];

    break;
  case 2:
    ff10[k] = f10[k];

    ff11[k] = f11[k] * nctop10[1] + f11[nv1[k]] * nctop11[1] +
        f11[nv3[k]] * nctop13[1] + f11[nv4[k]] * nctop14[1] +
        f11[nv7[k]] * nctop17[1] + f11[nv8[k]] * nctop18[1];

    ff12[k] = f12[k] * nctop10[2] + f12[nv1[k]] * nctop11[2] +
        f12[nv3[k]] * nctop13[2] + f12[nv4[k]] * nctop14[2] +
```

```
          f12[nv7[k]] * nctop17[2] + f12[nv8[k]] * nctop18[2];

     ff13[k] = f13[k] * nctop10[3] + f13[nv1[k]] * nctop11[3] +
          f13[nv3[k]] * nctop13[3] + f13[nv4[k]] * nctop14[3] +
          f13[nv7[k]] * nctop17[3] + f13[nv8[k]] * nctop18[3];

     ff14[k] = f14[k] * nctop10[4] + f14[nv1[k]] * nctop11[4] +
          f14[nv3[k]] * nctop13[4] + f14[nv4[k]] * nctop14[4] +
          f14[nv7[k]] * nctop17[4] + f14[nv8[k]] * nctop18[4];

     ff15[k] = f15[k] * nctop10[5] + f15[nv1[k]] * nctop11[5] +
          f15[nv3[k]] * nctop13[5] + f15[nv4[k]] * nctop14[5] +
          f15[nv7[k]] * nctop17[5] + f15[nv8[k]] * nctop18[5];

     ff16[k] = f16[k] * nctop10[6] + f16[nv1[k]] * nctop11[6] +
          f16[nv3[k]] * nctop13[6] + f16[nv4[k]] * nctop14[6] +
          f16[nv7[k]] * nctop17[6] + f16[nv8[k]] * nctop18[6];

     ff17[k] = f17[k] * nctop10[7] + f17[nv1[k]] * nctop11[7] +
          f17[nv3[k]] * nctop13[7] + f17[nv4[k]] * nctop14[7] +
          f17[nv7[k]] * nctop17[7] + f17[nv8[k]] * nctop18[7];

     ff18[k] = f18[k] * nctop10[8] + f18[nv1[k]] * nctop11[8] +
          f18[nv3[k]] * nctop13[8] + f18[nv4[k]] * nctop14[8] +
          f18[nv7[k]] * nctop17[8] + f18[nv8[k]] * nctop18[8];

     ff20[k] = f20[k];

     ff21[k] = f21[k] * nctop20[1] + f21[nv1[k]] * nctop21[1] +
          f21[nv3[k]] * nctop23[1] + f21[nv4[k]] * nctop24[1] +
          f21[nv7[k]] * nctop27[1] + f21[nv8[k]] * nctop28[1];

     ff22[k] = f22[k] * nctop20[2] + f22[nv1[k]] * nctop21[2] +
          f22[nv3[k]] * nctop23[2] + f22[nv4[k]] * nctop24[2] +
          f22[nv7[k]] * nctop27[2] + f22[nv8[k]] * nctop28[2];

     ff23[k] = f23[k] * nctop20[3] + f23[nv1[k]] * nctop21[3] +
          f23[nv3[k]] * nctop23[3] + f23[nv4[k]] * nctop24[3] +
          f23[nv7[k]] * nctop27[3] + f23[nv8[k]] * nctop28[3];

     ff24[k] = f24[k] * nctop20[4] + f24[nv1[k]] * nctop21[4] +
          f24[nv3[k]] * nctop23[4] + f24[nv4[k]] * nctop24[4] +
          f24[nv7[k]] * nctop27[4] + f24[nv8[k]] * nctop28[4];

     ff25[k] = f25[k] * nctop20[5] + f25[nv1[k]] * nctop21[5] +
          f25[nv3[k]] * nctop23[5] + f25[nv4[k]] * nctop24[5] +
          f25[nv7[k]] * nctop27[5] + f25[nv8[k]] * nctop28[5];

     ff26[k] = f26[k] * nctop20[6] + f26[nv1[k]] * nctop21[6] +
          f26[nv3[k]] * nctop23[6] + f26[nv4[k]] * nctop24[6] +
          f26[nv7[k]] * nctop27[6] + f26[nv8[k]] * nctop28[6];

     ff27[k] = f27[k] * nctop20[7] + f27[nv1[k]] * nctop21[7] +
          f27[nv3[k]] * nctop23[7] + f27[nv4[k]] * nctop24[7] +
          f27[nv7[k]] * nctop27[7] + f27[nv8[k]] * nctop28[7];

     ff28[k] = f28[k] * nctop20[8] + f28[nv1[k]] * nctop21[8] +
          f28[nv3[k]] * nctop23[8] + f28[nv4[k]] * nctop24[8] +
   f28[nv7[k]] * nctop27[8] + f28[nv8[k]] * nctop28[8];

     break;
   case 3:
     ff10[k] = f10[k];

     ff11[k] = f11[k] * ncleft10[1] + f11[nv1[k]] * ncleft11[1] +
          f11[nv5[k]] * ncleft15[1] + f11[nv2[k]] * ncleft12[1] +
          f11[nv4[k]] * ncleft14[1] + f11[nv8[k]] * ncleft18[1];

     ff12[k] = f12[k] * ncleft10[2] + f12[nv1[k]] * ncleft11[2] +
```



```
          f12[nv5[k]] * ncleft15[2] + f12[nv2[k]] * ncleft12[2] +
          f12[nv4[k]] * ncleft14[2] + f12[nv8[k]] * ncleft18[2];

     ff13[k] = f13[k] * ncleft10[3] + f13[nv1[k]] * ncleft11[3] +
          f13[nv5[k]] * ncleft15[3] + f13[nv2[k]] * ncleft12[3] +
          f13[nv4[k]] * ncleft14[3] + f13[nv8[k]] * ncleft18[3];

     ff14[k] = f14[k] * ncleft10[4] + f14[nv1[k]] * ncleft11[4] +
          f14[nv5[k]] * ncleft15[4] + f14[nv2[k]] * ncleft12[4] +
          f14[nv4[k]] * ncleft14[4] + f14[nv8[k]] * ncleft18[4];

     ff15[k] = f15[k] * ncleft10[5] + f15[nv1[k]] * ncleft11[5] +
          f15[nv5[k]] * ncleft15[5] + f15[nv2[k]] * ncleft12[5] +
          f15[nv4[k]] * ncleft14[5] + f15[nv8[k]] * ncleft18[5];

     ff16[k] = f16[k] * ncleft10[6] + f16[nv1[k]] * ncleft11[6] +
          f16[nv5[k]] * ncleft15[6] + f16[nv2[k]] * ncleft12[6] +
          f16[nv4[k]] * ncleft14[6] + f16[nv8[k]] * ncleft18[6];

     ff17[k] = f17[k] * ncleft10[7] + f17[nv1[k]] * ncleft11[7] +
          f17[nv5[k]] * ncleft15[7] + f17[nv2[k]] * ncleft12[7] +
          f17[nv4[k]] * ncleft14[7] + f17[nv8[k]] * ncleft18[7];

     ff18[k] = f18[k] * ncleft10[8] + f18[nv1[k]] * ncleft11[8] +
          f18[nv5[k]] * ncleft15[8] + f18[nv2[k]] * ncleft12[8] +
          f18[nv4[k]] * ncleft14[8] + f18[nv8[k]] * ncleft18[8];

     ff20[k] = f20[k];

     ff21[k] = f21[k] * ncleft20[1] + f21[nv1[k]] * ncleft21[1] +
          f21[nv5[k]] * ncleft25[1] + f21[nv2[k]] * ncleft22[1] +
          f21[nv4[k]] * ncleft24[1] + f21[nv8[k]] * ncleft28[1];

     ff22[k] = f22[k] * ncleft20[2] + f22[nv1[k]] * ncleft21[2] +
          f22[nv5[k]] * ncleft25[2] + f22[nv2[k]] * ncleft22[2] +
          f22[nv4[k]] * ncleft24[2] + f22[nv8[k]] * ncleft28[2];

     ff23[k] = f23[k] * ncleft20[3] + f23[nv1[k]] * ncleft21[3] +
          f23[nv5[k]] * ncleft25[3] + f23[nv2[k]] * ncleft22[3] +
          f23[nv4[k]] * ncleft24[3] + f23[nv8[k]] * ncleft28[3];

     ff24[k] = f24[k] * ncleft20[4] + f24[nv1[k]] * ncleft21[4] +
          f24[nv5[k]] * ncleft25[4] + f24[nv2[k]] * ncleft22[4] +
          f24[nv4[k]] * ncleft24[4] + f24[nv8[k]] * ncleft28[4];

     ff25[k] = f25[k] * ncleft20[5] + f25[nv1[k]] * ncleft21[5] +
          f25[nv5[k]] * ncleft25[5] + f25[nv2[k]] * ncleft22[5] +
          f25[nv4[k]] * ncleft24[5] + f25[nv8[k]] * ncleft28[5];

     ff26[k] = f26[k] * ncleft20[6] + f26[nv1[k]] * ncleft21[6] +
          f26[nv5[k]] * ncleft25[6] + f26[nv2[k]] * ncleft22[6] +
          f26[nv4[k]] * ncleft24[6] + f26[nv8[k]] * ncleft28[6];

     ff27[k] = f27[k] * ncleft20[7] + f27[nv1[k]] * ncleft21[7] +
          f27[nv5[k]] * ncleft25[7] + f27[nv2[k]] * ncleft22[7] +
          f27[nv4[k]] * ncleft24[7] + f27[nv8[k]] * ncleft28[7];

     ff28[k] = f28[k] * ncleft20[8] + f28[nv1[k]] * ncleft21[8] +
          f28[nv5[k]] * ncleft25[8] + f28[nv2[k]] * ncleft22[8] +
          f28[nv4[k]] * ncleft24[8] + f28[nv8[k]] * ncleft28[8];

     break;
   case 4:
     ff10[k] = f10[k];

     ff11[k] = f11[k] * ncright10[1] + f11[nv3[k]] * ncright13[1] +
          f11[nv6[k]] * ncright16[1] + f11[nv2[k]] * ncright12[1] +
          f11[nv4[k]] * ncright14[1] + f11[nv7[k]] * ncright17[1];
```

```
        ff12[k] = f12[k] * ncright10[2] + f12[nv3[k]] * ncright13[2] +
              f12[nv6[k]] * ncright16[2] + f12[nv2[k]] * ncright12[2] +
              f12[nv4[k]] * ncright14[2] + f12[nv7[k]] * ncright17[2];

        ff13[k] = f13[k] * ncright10[3] + f13[nv3[k]] * ncright13[3] +
              f13[nv6[k]] * ncright16[3] + f13[nv2[k]] * ncright12[3] +
              f13[nv4[k]] * ncright14[3] + f13[nv7[k]] * ncright17[3];

        ff14[k] = f14[k] * ncright10[4] + f14[nv3[k]] * ncright13[4] +
              f14[nv6[k]] * ncright16[4] + f14[nv2[k]] * ncright12[4] +
              f14[nv4[k]] * ncright14[4] + f14[nv7[k]] * ncright17[4];

        ff15[k] = f15[k] * ncright10[5] + f15[nv3[k]] * ncright13[5] +
              f15[nv6[k]] * ncright16[5] + f15[nv2[k]] * ncright12[5] +
              f15[nv4[k]] * ncright14[5] + f15[nv7[k]] * ncright17[5];

        ff16[k] = f16[k] * ncright10[6] + f16[nv3[k]] * ncright13[6] +
              f16[nv6[k]] * ncright16[6] + f16[nv2[k]] * ncright12[6] +
              f16[nv4[k]] * ncright14[6] + f16[nv7[k]] * ncright17[6];

        ff17[k] = f17[k] * ncright10[7] + f17[nv3[k]] * ncright13[7] +
              f17[nv6[k]] * ncright16[7] + f17[nv2[k]] * ncright12[7] +
              f17[nv4[k]] * ncright14[7] + f17[nv7[k]] * ncright17[7];

        ff18[k] = f18[k] * ncright10[8] + f18[nv3[k]] * ncright13[8] +
              f18[nv6[k]] * ncright16[8] + f18[nv2[k]] * ncright12[8] +
              f18[nv4[k]] * ncright14[8] + f18[nv7[k]] * ncright17[8];

        ff20[k] = f20[k];

        ff21[k] = f21[k] * ncright20[1] + f21[nv3[k]] * ncright23[1] +
              f21[nv6[k]] * ncright26[1] + f21[nv2[k]] * ncright22[1] +
              f21[nv4[k]] * ncright24[1] + f21[nv7[k]] * ncright27[1];

        ff22[k] = f22[k] * ncright20[2] + f22[nv3[k]] * ncright23[2] +
              f22[nv6[k]] * ncright26[2] + f22[nv2[k]] * ncright22[2] +
              f22[nv4[k]] * ncright24[2] + f22[nv7[k]] * ncright27[2];

        ff23[k] = f23[k] * ncright20[3] + f23[nv3[k]] * ncright23[3] +
              f23[nv6[k]] * ncright26[3] + f23[nv2[k]] * ncright22[3] +
              f23[nv4[k]] * ncright24[3] + f23[nv7[k]] * ncright27[3];

        ff24[k] = f24[k] * ncright20[4] + f24[nv3[k]] * ncright23[4] +
              f24[nv6[k]] * ncright26[4] + f24[nv2[k]] * ncright22[4] +
              f24[nv4[k]] * ncright24[4] + f24[nv7[k]] * ncright27[4];

        ff25[k] = f25[k] * ncright20[5] + f25[nv3[k]] * ncright23[5] +
              f25[nv6[k]] * ncright26[5] + f25[nv2[k]] * ncright22[5] +
              f25[nv4[k]] * ncright24[5] + f25[nv7[k]] * ncright27[5];

        ff26[k] = f26[k] * ncright20[6] + f26[nv3[k]] * ncright23[6] +
              f26[nv6[k]] * ncright26[6] + f26[nv2[k]] * ncright22[6] +
              f26[nv4[k]] * ncright24[6] + f26[nv7[k]] * ncright27[6];

        ff27[k] = f27[k] * ncright20[7] + f27[nv3[k]] * ncright23[7] +
              f27[nv6[k]] * ncright26[7] + f27[nv2[k]] * ncright22[7] +
              f27[nv4[k]] * ncright24[7] + f27[nv7[k]] * ncright27[7];

        ff28[k] = f28[k] * ncright20[8] + f28[nv3[k]] * ncright23[8] +
              f28[nv6[k]] * ncright26[8] + f28[nv2[k]] * ncright22[8] +
              f28[nv4[k]] * ncright24[8] + f28[nv7[k]] * ncright27[8];

        break;        }
    }
}

void ifdprop(void)
{
```



```
   int k;
   double prodscal;

   for(k=0; k<nnodes_all; k++)
     {
       prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k];

       f10[k] = ff10[k] - ctau1 * ( ff10[k] - neq10[k] ) +
                (ecprod1[0] - prodscal) * neq10[k];
       f11[k] = ff11[k] - ctau1 * ( ff11[k] - neq11[k] ) +
                (ecprod1[1] - prodscal) * neq11[k];
       f12[k] = ff12[k] - ctau1 * ( ff12[k] - neq12[k] ) +
                (ecprod1[2] - prodscal) * neq12[k];
       f13[k] = ff13[k] - ctau1 * ( ff13[k] - neq13[k] ) +
                (ecprod1[3] - prodscal) * neq13[k];
       f14[k] = ff14[k] - ctau1 * ( ff14[k] - neq14[k] ) +
                (ecprod1[4] - prodscal) * neq14[k];
       f15[k] = ff15[k] - ctau1 * ( ff15[k] - neq15[k] ) +
                (ecprod1[5] - prodscal) * neq15[k];
       f16[k] = ff16[k] - ctau1 * ( ff16[k] - neq16[k] ) +
                (ecprod1[6] - prodscal) * neq16[k];
       f17[k] = ff17[k] - ctau1 * ( ff17[k] - neq17[k] ) +
                (ecprod1[7] - prodscal) * neq17[k];
       f18[k] = ff18[k] - ctau1 * ( ff18[k] - neq18[k] ) +
                (ecprod1[8] - prodscal) * neq18[k];

       f20[k] = ff20[k] - ctau2 * ( ff20[k] - neq20[k] ) +
                (ecprod2[0] - prodscal) * neq20[k];
       f21[k] = ff21[k] - ctau2 * ( ff21[k] - neq21[k] ) +
                (ecprod2[1] - prodscal) * neq21[k];
       f22[k] = ff22[k] - ctau2 * ( ff22[k] - neq22[k] ) +
                (ecprod2[2] - prodscal) * neq22[k];
       f23[k] = ff23[k] - ctau2 * ( ff23[k] - neq23[k] ) +
                (ecprod2[3] - prodscal) * neq23[k];
       f24[k] = ff24[k] - ctau2 * ( ff24[k] - neq24[k] ) +
                (ecprod2[4] - prodscal) * neq24[k];
       f25[k] = ff25[k] - ctau2 * ( ff25[k] - neq25[k] ) +
                (ecprod2[5] - prodscal) * neq25[k];
       f26[k] = ff26[k] - ctau2 * ( ff26[k] - neq26[k] ) +
                (ecprod2[6] - prodscal) * neq26[k];
       f27[k] = ff27[k] - ctau2 * ( ff27[k] - neq27[k] ) +
                (ecprod2[7] - prodscal) * neq27[k];
       f28[k] = ff28[k] - ctau2 * ( ff28[k] - neq28[k] ) +
                (ecprod2[8] - prodscal) * neq28[k];

       /*
       f10[k] = ff10[k] - ctau1 * ( f10[k] - neq10[k] ) +
                (ecprod1[0] - prodscal) * neq10[k];
       f11[k] = ff11[k] - ctau1 * ( f11[k] - neq11[k] ) +
                (ecprod1[1] - prodscal) * neq11[k];
       f12[k] = ff12[k] - ctau1 * ( f12[k] - neq12[k] ) +
                (ecprod1[2] - prodscal) * neq12[k];
       f13[k] = ff13[k] - ctau1 * ( f13[k] - neq13[k] ) +
                (ecprod1[3] - prodscal) * neq13[k];
       f14[k] = ff14[k] - ctau1 * ( f14[k] - neq14[k] ) +
                (ecprod1[4] - prodscal) * neq14[k];
       f15[k] = ff15[k] - ctau1 * ( f15[k] - neq15[k] ) +
                (ecprod1[5] - prodscal) * neq15[k];
       f16[k] = ff16[k] - ctau1 * ( f16[k] - neq16[k] ) +
                (ecprod1[6] - prodscal) * neq16[k];
       f17[k] = ff17[k] - ctau1 * ( f17[k] - neq17[k] ) +
                (ecprod1[7] - prodscal) * neq17[k];
       f18[k] = ff18[k] - ctau1 * ( f18[k] - neq18[k] ) +
                (ecprod1[8] - prodscal) * neq18[k];

       f20[k] = ff20[k] - ctau2 * ( f20[k] - neq20[k] ) +
                (ecprod2[0] - prodscal) * neq20[k];
       f21[k] = ff21[k] - ctau2 * ( f21[k] - neq21[k] ) +
                (ecprod2[1] - prodscal) * neq21[k];
```

```
      f22[k] = ff22[k] - ctau2 * ( f22[k] - neq22[k] ) +
               (ecprod2[2] - prodscal) * neq22[k];
      f23[k] = ff23[k] - ctau2 * ( f23[k] - neq23[k] ) +
               (ecprod2[3] - prodscal) * neq23[k];
      f24[k] = ff24[k] - ctau2 * ( f24[k] - neq24[k] ) +
               (ecprod2[4] - prodscal) * neq24[k];
      f25[k] = ff25[k] - ctau2 * ( f25[k] - neq25[k] ) +
               (ecprod2[5] - prodscal) * neq25[k];
      f26[k] = ff26[k] - ctau2 * ( f26[k] - neq26[k] ) +
               (ecprod2[6] - prodscal) * neq26[k];
      f27[k] = ff27[k] - ctau2 * ( f27[k] - neq27[k] ) +
               (ecprod2[7] - prodscal) * neq27[k];
      f28[k] = ff28[k] - ctau2 * ( f28[k] - neq28[k] ) +
               (ecprod2[8] - prodscal) * neq28[k];
      */
      /*
      if(k==0)
        printf("iter=%d  k=%d  ff10=%e  ff20=%e\n",iter,k,ff10[k],ff20[k]);
        printf("iter=%d  k=%d  neq10=%e  f10-neq=%e\n",iter,k,neq10[k],
               f10[k]-neq10[k]);
      */

    }
}

void iupfd(void)
{
  int k;
  for(k=0; k<nnodes_all; k++)
    {
      ff10[k] = f10[k];

      ff11[k] = f11[k] * nu11[1] +
           f11[nv3[k]] * nu10[1] + f11[nv3[nv3[k]]] * nu13[1] +
           f11[nv2[k]] * nu15[1] +       f11[nv6[k]] * nu12[1] +
      f11[nv3[nv6[k]]] * nu16[1] +
           f11[nv4[k]] * nu18[1] +       f11[nv7[k]] * nu14[1] +
      f11[nv3[nv7[k]]] * nu17[1];

      ff12[k] = f12[k] * nu12[2] +
           f12[nv4[k]] * nu10[2] + f12[nv4[nv4[k]]] * nu14[2] +
           f12[nv1[k]] * nu15[2] +       f12[nv8[k]] * nu11[2] +
      f12[nv4[nv8[k]]] * nu18[2] +
           f12[nv3[k]] * nu16[2] +
           f12[nv7[k]] * nu13[2] + f12[nv4[nv7[k]]] * nu17[2];

      ff13[k] = f13[k] * nu13[3] +
           f13[nv1[k]] * nu10[3] + f13[nv1[nv1[k]]] * nu11[3] +
           f13[nv4[k]] * nu17[3] +       f13[nv8[k]] * nu14[3] +
      f13[nv1[nv8[k]]] * nu18[3] +
           f13[nv2[k]] * nu16[3] +
           f13[nv5[k]] * nu12[3] + f13[nv1[nv5[k]]] * nu15[3];

      ff14[k] = f14[k] * nu14[4] +
           f14[nv2[k]] * nu10[4] + f14[nv2[nv2[k]]] * nu12[4] +
           f14[nv1[k]] * nu18[4] +       f14[nv5[k]] * nu11[4] +
      f14[nv2[nv5[k]]] * nu15[4] +
           f14[nv3[k]] * nu17[4] +
           f14[nv5[k]] * nu13[4] + f14[nv2[nv5[k]]] * nu16[4];

      ff15[k] = f15[k] * nu15[5] +
           f15[nv3[k]] * nu12[5] + f15[nv3[nv3[k]]] * nu16[5] +
           f15[nv4[k]] * nu11[5] +       f15[nv7[k]] * nu10[5] +
      f15[nv3[nv7[k]]] * nu13[5] + f15[nv4[nv4[k]]] * nu18[5] +
      f15[nv4[nv7[k]]] * nu14[5] + f15[nv7[nv7[k]]] * nu17[5];

      ff16[k] = f16[k] * nu16[6] +
           f16[nv1[k]] * nu12[6] + f16[nv1[nv1[k]]] * nu15[6] +
           f16[nv4[k]] * nu13[6] +       f16[nv8[k]] * nu10[6] +
```



```
      f16[nv1[nv8[k]]] * nu11[6] + f16[nv4[nv4[k]]] * nu17[6] +
      f16[nv4[nv8[k]]] * nu14[6] + f16[nv8[nv8[k]]] * nu18[6];

      ff17[k] = f17[k] * nu17[7] +
           f17[nv1[k]] * nu14[7] + f17[nv1[nv1[k]]] * nu18[7] +
           f17[nv2[k]] * nu13[7] +       f17[nv5[k]] * nu10[7] +
      f17[nv1[nv5[k]]] * nu11[7] + f17[nv2[nv2[k]]] * nu16[7] +
      f17[nv2[nv5[k]]] * nu12[7] + f17[nv5[nv5[k]]] * nu15[7];

      ff18[k] = f18[k] * nu18[8] +
           f18[nv3[k]] * nu14[8] + f18[nv3[nv3[k]]] * nu17[8] +
           f18[nv2[k]] * nu11[8] +       f18[nv6[k]] * nu10[8] +
      f18[nv3[nv6[k]]] * nu13[8] + f18[nv2[nv2[k]]] * nu15[8] +
      f18[nv2[nv6[k]]] * nu12[8] + f18[nv6[nv6[k]]] * nu16[8];

      ff20[k] = f20[k];

      ff21[k] = f21[k] * nu21[1] +
           f21[nv3[k]] * nu20[1] + f21[nv3[nv3[k]]] * nu23[1] +
           f21[nv2[k]] * nu25[1] +       f21[nv6[k]] * nu22[1] +
      f21[nv3[nv6[k]]] * nu26[1] +
           f21[nv4[k]] * nu28[1] +       f21[nv7[k]] * nu24[1] +
      f21[nv3[nv7[k]]] * nu27[1];

      ff22[k] = f22[k] * nu22[2] +
           f22[nv4[k]] * nu20[2] + f22[nv4[nv4[k]]] * nu24[2] +
           f22[nv1[k]] * nu25[2] +       f22[nv8[k]] * nu21[2] +
      f22[nv4[nv8[k]]] * nu28[2] +
           f22[nv3[k]] * nu26[2] +
           f22[nv7[k]] * nu23[2] + f22[nv4[nv7[k]]] * nu27[2];

      ff23[k] = f23[k] * nu23[3] +
           f23[nv1[k]] * nu20[3] + f23[nv1[nv1[k]]] * nu21[3] +
           f23[nv4[k]] * nu27[3] +       f23[nv8[k]] * nu24[3] +
      f23[nv1[nv8[k]]] * nu28[3] +
           f23[nv2[k]] * nu26[3] +
           f23[nv5[k]] * nu22[3] + f23[nv1[nv5[k]]] * nu25[3];

      ff24[k] = f24[k] * nu24[4] +
           f24[nv2[k]] * nu20[4] + f24[nv2[nv2[k]]] * nu22[4] +
           f24[nv1[k]] * nu28[4] +       f24[nv5[k]] * nu21[4] +
      f24[nv2[nv5[k]]] * nu25[4] +
           f24[nv3[k]] * nu27[4] +
           f24[nv5[k]] * nu23[4] + f24[nv2[nv5[k]]] * nu26[4];

      ff25[k] = f25[k] * nu25[5] +
           f25[nv3[k]] * nu22[5] + f25[nv3[nv3[k]]] * nu26[5] +
           f25[nv4[k]] * nu21[5] +       f25[nv7[k]] * nu20[5] +
      f25[nv3[nv7[k]]] * nu23[5] + f25[nv4[nv4[k]]] * nu28[5] +
      f25[nv4[nv7[k]]] * nu24[5] + f25[nv7[nv7[k]]] * nu27[5];

      ff26[k] = f26[k] * nu26[6] +
           f26[nv1[k]] * nu22[6] + f26[nv1[nv1[k]]] * nu25[6] +
           f26[nv4[k]] * nu23[6] +       f26[nv8[k]] * nu20[6] +
      f26[nv1[nv8[k]]] * nu21[6] + f26[nv4[nv4[k]]] * nu27[6] +
      f26[nv4[nv8[k]]] * nu24[6] + f26[nv8[nv8[k]]] * nu28[6];

      ff27[k] = f27[k] * nu27[7] +
           f27[nv1[k]] * nu24[7] + f27[nv1[nv1[k]]] * nu28[7] +
           f27[nv2[k]] * nu23[7] +       f27[nv5[k]] * nu20[7] +
      f27[nv1[nv5[k]]] * nu21[7] + f27[nv2[nv2[k]]] * nu26[7] +
      f27[nv2[nv5[k]]] * nu22[7] + f27[nv5[nv5[k]]] * nu25[7];

      ff28[k] = f28[k] * nu28[8] +
           f28[nv3[k]] * nu24[8] + f28[nv3[nv3[k]]] * nu27[8] +
           f28[nv2[k]] * nu21[8] +       f28[nv6[k]] * nu20[8] +
      f28[nv3[nv6[k]]] * nu23[8] + f28[nv2[nv2[k]]] * nu25[8] +
      f28[nv2[nv6[k]]] * nu22[8] + f28[nv6[nv6[k]]] * nu26[8];
```

```c
      /*
      if(k==0)
        printf("iter=%d  k=%d  f1=%e  f2=%e\n",iter,k,f10[k],f20[k]);
      */
    }
}

void iser(void)
{
  int k;
  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
         case 0:
           ff10[k] = f10[k];

           ff11[k] =     f11[k] * ns1tr[1] + f11[nv3[k]] * ns1tl[1] +
                     f11[nv7[k]] * ns1bl[1] + f11[nv4[k]] * ns1br[1];

           ff12[k] =     f12[k] * ns1tl[2] + f12[nv4[k]] * ns1bl[2] +
                     f12[nv1[k]] * ns1tr[2] + f12[nv8[k]] * ns1br[2];

           ff13[k] =     f13[k] * ns1bl[3] + f13[nv1[k]] * ns1br[3] +
                     f13[nv2[k]] * ns1tl[3] + f13[nv5[k]] * ns1tr[3];

           ff14[k] =     f14[k] * ns1br[4] + f14[nv3[k]] * ns1bl[4] +
                     f14[nv2[k]] * ns1tr[4] + f14[nv6[k]] * ns1tl[4];

           ff15[k] =     f15[k] * ns1tr[5] + f15[nv3[k]] * ns1tl[5] +
                     f15[nv7[k]] * ns1bl[5] + f15[nv4[k]] * ns1br[5];

           ff16[k] =     f16[k] * ns1tl[6] + f16[nv4[k]] * ns1bl[6] +
                     f16[nv1[k]] * ns1tr[6] + f16[nv8[k]] * ns1br[6];

           ff17[k] =     f17[k] * ns1bl[7] + f17[nv1[k]] * ns1br[7] +
                     f17[nv2[k]] * ns1tl[7] + f17[nv5[k]] * ns1tr[7];

           ff18[k] =     f18[k] * ns1br[8] + f18[nv3[k]] * ns1bl[8] +
                     f18[nv2[k]] * ns1tr[8] + f18[nv6[k]] * ns1tl[8];

           ff20[k] = f20[k];

           ff21[k] =     f21[k] * ns2tr[1] + f21[nv3[k]] * ns2tl[1] +
                     f21[nv7[k]] * ns2bl[1] + f21[nv4[k]] * ns2br[1];

           ff22[k] =     f22[k] * ns2tl[2] + f22[nv4[k]] * ns2bl[2] +
                     f22[nv1[k]] * ns2tr[2] + f22[nv8[k]] * ns2br[2];

           ff23[k] =     f23[k] * ns2bl[3] + f23[nv1[k]] * ns2br[3] +
                     f23[nv2[k]] * ns2tl[3] + f23[nv5[k]] * ns2tr[3];

           ff24[k] =     f24[k] * ns2br[4] + f24[nv3[k]] * ns2bl[4] +
                     f24[nv2[k]] * ns2tr[4] + f24[nv6[k]] * ns2tl[4];

           ff25[k] =     f25[k] * ns2tr[5] + f25[nv3[k]] * ns2tl[5] +
                     f25[nv7[k]] * ns2bl[5] + f25[nv4[k]] * ns2br[5];

           ff26[k] =     f26[k] * ns2tl[6] + f26[nv4[k]] * ns2bl[6] +
                     f26[nv1[k]] * ns2tr[6] + f26[nv8[k]] * ns2br[6];

           ff27[k] =     f27[k] * ns2bl[7] + f27[nv1[k]] * ns2br[7] +
                     f27[nv2[k]] * ns2tl[7] + f27[nv5[k]] * ns2tr[7];

           ff28[k] =     f28[k] * ns2br[8] + f28[nv3[k]] * ns2bl[8] +
                     f28[nv2[k]] * ns2tr[8] + f28[nv6[k]] * ns2tl[8];
           break;

         case 1:
```



```c
           ff10[k] = f10[k];

           ff11[k] =     f11[k] * nsb1br[1] + f11[nv3[k]] * nsb1bl[1] +
                     f11[nv6[k]] * nsb1tl[1] + f11[nv2[k]] * nsb1tr[1];

           ff12[k] =     f12[k] * nsb1bl[2] + f12[nv1[k]] * nsb1br[2] +
                     f12[nv2[k]] * nsb1tl[2] + f12[nv5[k]] * nsb1tr[2];

           ff13[k] =     f13[k] * ns1bl[3] + f13[nv1[k]] * ns1br[3] +
                     f13[nv2[k]] * ns1tl[3] + f13[nv5[k]] * ns1tr[3];

           ff14[k] =     f14[k] * ns1br[4] + f14[nv3[k]] * ns1bl[4] +
                     f14[nv2[k]] * ns1tr[4] + f14[nv6[k]] * ns1tl[4];

           ff15[k] =     f15[k] * nsb1bl[5] + f15[nv1[k]] * nsb1br[5] +
                     f15[nv2[k]] * nsb1tl[5] + f15[nv5[k]] * nsb1tr[5];

           ff16[k] =     f16[k] * nsb1br[6] + f16[nv3[k]] * nsb1bl[6] +
                     f16[nv2[k]] * nsb1tr[6] + f16[nv6[k]] * nsb1tl[6];

           ff17[k] =     f17[k] * ns1bl[7] + f17[nv1[k]] * ns1br[7] +
                     f17[nv2[k]] * ns1tl[7] + f17[nv5[k]] * ns1tr[7];

           ff18[k] =     f18[k] * ns1br[8] + f18[nv3[k]] * ns1bl[8] +
                     f18[nv2[k]] * ns1tr[8] + f18[nv6[k]] * ns1tl[8];

           ff20[k] = f20[k];

           ff21[k] =     f21[k] * nsb2br[1] + f21[nv3[k]] * nsb2bl[1] +
                     f21[nv6[k]] * nsb2tl[1] + f21[nv2[k]] * nsb2tr[1];

           ff22[k] =     f22[k] * nsb2bl[2] + f22[nv1[k]] * nsb2br[2] +
                     f22[nv2[k]] * nsb2tl[2] + f22[nv5[k]] * nsb2tr[2];

           ff23[k] =     f23[k] * ns2bl[3] + f23[nv1[k]] * ns2br[3] +
                     f23[nv2[k]] * ns2tl[3] + f23[nv5[k]] * ns2tr[3];

           ff24[k] =     f24[k] * ns2br[4] + f24[nv3[k]] * ns2bl[4] +
                     f24[nv2[k]] * ns2tr[4] + f24[nv6[k]] * ns2tl[4];

           ff25[k] =     f25[k] * nsb2bl[5] + f25[nv1[k]] * nsb2br[5] +
                     f25[nv2[k]] * nsb2tl[5] + f25[nv5[k]] * nsb2tr[5];

           ff26[k] =     f26[k] * nsb2br[6] + f26[nv3[k]] * nsb2bl[6] +
                     f26[nv2[k]] * nsb2tr[6] + f26[nv6[k]] * nsb2tl[6];

           ff27[k] =     f27[k] * ns2bl[7] + f27[nv1[k]] * ns2br[7] +
                     f27[nv2[k]] * ns2tl[7] + f27[nv5[k]] * ns2tr[7];

           ff28[k] =     f28[k] * ns2br[8] + f28[nv3[k]] * ns2bl[8] +
                     f28[nv2[k]] * ns2tr[8] + f28[nv6[k]] * ns2tl[8];
           break;

         case 2:
           ff10[k] = f10[k];

           ff11[k] =     f11[k] * ns1tr[1] + f11[nv3[k]] * ns1tl[1] +
                     f11[nv7[k]] * ns1bl[1] + f11[nv4[k]] * ns1br[1];

           ff12[k] =     f12[k] * ns1tl[2] + f12[nv4[k]] * ns1bl[2] +
                     f12[nv1[k]] * ns1tr[2] + f12[nv8[k]] * ns1br[2];

           ff13[k] =     f13[k] * nst1tl[3] + f13[nv1[k]] * nst1tr[3] +
                     f13[nv4[k]] * nst1bl[3] + f13[nv8[k]] * nst1br[3];

           ff14[k] =     f14[k] * nst1tr[4] + f14[nv3[k]] * nst1tl[4] +
                     f14[nv4[k]] * nst1br[4] + f14[nv7[k]] * nst1bl[4];

           ff15[k] =     f15[k] * ns1tr[5] + f15[nv3[k]] * ns1tl[5] +
```

```
            f15[nv7[k]] * ns1bl[5] + f15[nv4[k]] * ns1br[5];

   ff16[k] =     f16[k] * ns1tl[6] + f16[nv4[k]] * ns1bl[6] +
            f16[nv1[k]] * ns1tr[6] + f16[nv8[k]] * ns1br[6];

   ff17[k] =     f17[k] * nst1tr[7] + f17[nv3[k]] * nst1tl[7] +
            f17[nv4[k]] * nst1br[7] + f17[nv7[k]] * nst1bl[7];

   ff18[k] =     f18[k] * nst1tl[8] + f18[nv1[k]] * nst1tr[8] +
            f18[nv4[k]] * nst1bl[8] + f18[nv8[k]] * nst1br[8];

   ff20[k] = f20[k];

   ff21[k] =     f21[k] * ns2tr[1] + f21[nv3[k]] * ns2tl[1] +
            f21[nv7[k]] * ns2bl[1] + f21[nv4[k]] * ns2br[1];

   ff22[k] =     f22[k] * ns2tl[2] + f22[nv4[k]] * ns2bl[2] +
            f22[nv1[k]] * ns2tr[2] + f22[nv8[k]] * ns2br[2];

   ff23[k] =     f23[k] * nst2tl[3] + f23[nv1[k]] * nst2tr[3] +
            f23[nv4[k]] * nst2bl[3] + f23[nv8[k]] * nst2br[3];

   ff24[k] =     f24[k] * nst2tr[4] + f24[nv3[k]] * nst2tl[4] +
            f24[nv4[k]] * nst2br[4] + f24[nv7[k]] * nst2bl[4];

   ff25[k] =     f25[k] * ns2tr[5] + f25[nv3[k]] * ns2tl[5] +
            f25[nv7[k]] * ns2bl[5] + f25[nv4[k]] * ns2br[5];

   ff26[k] =     f26[k] * ns2tl[6] + f26[nv4[k]] * ns2bl[6] +
            f26[nv1[k]] * ns2tr[6] + f26[nv8[k]] * ns2br[6];

   ff27[k] =     f27[k] * nst2tr[7] + f27[nv3[k]] * nst2tl[7] +
            f27[nv4[k]] * nst2br[7] + f27[nv7[k]] * nst2bl[7];

   ff28[k] =     f28[k] * nst2tl[8] + f28[nv1[k]] * nst2tr[8] +
            f28[nv4[k]] * nst2bl[8] + f28[nv8[k]] * nst2br[8];

   break;

  case 3:
   ff10[k] = f10[k];

   ff11[k] =     f11[k] * ns11tl[1] + f11[nv1[k]] * ns11tr[1] +
            f11[nv4[k]] * ns11bl[1] + f11[nv8[k]] * ns11br[1];

   ff12[k] =     f12[k] * ns1tl[2] + f12[nv4[k]] * ns1bl[2] +
            f12[nv1[k]] * ns1tr[2] + f12[nv8[k]] * ns1br[2];

   ff13[k] =     f13[k] * ns1bl[3] + f13[nv1[k]] * ns1br[3] +
            f13[nv2[k]] * ns1tl[3] + f13[nv5[k]] * ns1tr[3];

   ff14[k] =     f14[k] * ns11bl[4] + f14[nv1[k]] * ns11br[4] +
            f14[nv2[k]] * ns11tl[4] + f14[nv5[k]] * ns11tr[4];

   ff15[k] =     f15[k] * ns11bl[5] + f15[nv1[k]] * ns11br[5] +
            f15[nv2[k]] * ns11tl[5] + f15[nv5[k]] * ns11tr[5];

   ff16[k] =     f16[k] * ns1tl[6] + f16[nv4[k]] * ns1bl[6] +
            f16[nv1[k]] * ns1tr[6] + f16[nv8[k]] * ns1br[6];

   ff17[k] =     f17[k] * ns1bl[7] + f17[nv1[k]] * ns1br[7] +
            f17[nv2[k]] * ns1tl[7] + f17[nv5[k]] * ns1tr[7];

   ff18[k] =     f18[k] * ns11tl[8] + f18[nv1[k]] * ns11tr[8] +
            f18[nv4[k]] * ns11bl[8] + f18[nv8[k]] * ns11br[8];

   ff20[k] = f20[k];

   ff21[k] =     f21[k] * ns12tl[1] + f21[nv1[k]] * ns12tr[1] +
```



```
            f21[nv4[k]] * ns12bl[1] + f21[nv8[k]] * ns12br[1];

   ff22[k] =     f22[k] * ns2tl[2] + f22[nv4[k]] * ns2bl[2] +
            f22[nv1[k]] * ns2tr[2] + f22[nv8[k]] * ns2br[2];

   ff23[k] =     f23[k] * ns2bl[3] + f23[nv1[k]] * ns2br[3] +
            f23[nv2[k]] * ns2tl[3] + f23[nv5[k]] * ns2tr[3];

   ff24[k] =     f24[k] * ns12bl[4] + f24[nv1[k]] * ns12br[4] +
            f24[nv2[k]] * ns12tl[4] + f24[nv5[k]] * ns12tr[4];

   ff25[k] =     f25[k] * ns12bl[5] + f25[nv1[k]] * ns12br[5] +
            f25[nv2[k]] * ns12tl[5] + f25[nv5[k]] * ns12tr[5];

   ff26[k] =     f26[k] * ns2tl[6] + f26[nv4[k]] * ns2bl[6] +
            f26[nv1[k]] * ns2tr[6] + f26[nv8[k]] * ns2br[6];

   ff27[k] =     f27[k] * ns2bl[7] + f27[nv1[k]] * ns2br[7] +
            f27[nv2[k]] * ns2tl[7] + f27[nv5[k]] * ns2tr[7];

   ff28[k] =     f28[k] * ns12tl[8] + f28[nv1[k]] * ns12tr[8] +
            f28[nv4[k]] * ns12bl[8] + f28[nv8[k]] * ns12br[8];

   break;

  case 4:
   ff10[k] = f10[k];

   ff11[k] =     f11[k] * ns1tr[1] + f11[nv3[k]] * ns1tl[1] +
            f11[nv7[k]] * ns1bl[1] + f11[nv4[k]] * ns1br[1];

   ff12[k] =     f12[k] * nsr1tr[2] + f12[nv3[k]] * nsr1tl[2] +
            f12[nv4[k]] * nsr1br[2] + f12[nv7[k]] * nsr1bl[2];

   ff13[k] =     f13[k] * nsr1br[3] + f13[nv3[k]] * nsr1bl[3] +
            f13[nv2[k]] * nsr1tr[3] + f13[nv6[k]] * nsr1tl[3];

   ff14[k] =     f14[k] * ns1br[4] + f14[nv3[k]] * ns1bl[4] +
            f14[nv2[k]] * ns1tr[4] + f14[nv6[k]] * ns1tl[4];

   ff15[k] =     f15[k] * ns1tr[5] + f15[nv3[k]] * ns1tl[5] +
            f15[nv7[k]] * ns1bl[5] + f15[nv4[k]] * ns1br[5];

   ff16[k] =     f16[k] * nsr1br[6] + f16[nv3[k]] * nsr1bl[6] +
            f16[nv2[k]] * nsr1tr[6] + f16[nv6[k]] * nsr1tl[6];

   ff17[k] =     f17[k] * nsr1tr[7] + f17[nv3[k]] * nsr1tl[7] +
            f17[nv4[k]] * nsr1br[7] + f17[nv7[k]] * nsr1bl[7];

   ff18[k] =     f18[k] * ns1br[8] + f18[nv3[k]] * ns1bl[8] +
            f18[nv2[k]] * ns1tr[8] + f18[nv6[k]] * ns1tl[8];

   ff20[k] = f20[k];

   ff21[k] =     f21[k] * ns2tr[1] + f21[nv3[k]] * ns2tl[1] +
            f21[nv7[k]] * ns2bl[1] + f21[nv4[k]] * ns2br[1];

   ff22[k] =     f22[k] * nsr2tr[2] + f22[nv3[k]] * nsr2tl[2] +
            f22[nv4[k]] * nsr2br[2] + f22[nv7[k]] * nsr2bl[2];

   ff23[k] =     f23[k] * nsr2br[3] + f23[nv3[k]] * nsr2bl[3] +
            f23[nv2[k]] * nsr2tr[3] + f23[nv6[k]] * nsr2tl[3];

   ff24[k] =     f24[k] * ns2br[4] + f24[nv3[k]] * ns2bl[4] +
            f24[nv2[k]] * ns2tr[4] + f24[nv6[k]] * ns2tl[4];

   ff25[k] =     f25[k] * ns2tr[5] + f25[nv3[k]] * ns2tl[5] +
            f25[nv7[k]] * ns2bl[5] + f25[nv4[k]] * ns2br[5];
```

```
            ff26[k] =      f26[k] * nsr2br[6] + f26[nv3[k]] * nsr2bl[6] +
                      f26[nv2[k]] * nsr2tr[6] + f26[nv6[k]] * nsr2tl[6];

            ff27[k] =      f27[k] * nsr2tr[7] + f27[nv3[k]] * nsr2tl[7] +
                      f27[nv4[k]] * nsr2br[7] + f27[nv7[k]] * nsr2bl[7];

            ff28[k] =      f28[k] * ns2br[8] + f28[nv3[k]] * ns2bl[8] +
                      f28[nv2[k]] * ns2tr[8] + f28[nv6[k]] * ns2tl[8];
            break;
        }
    }
}


void ilin(void)
{
  int k;
  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:
            ff10[k] = f10[k];

            ff11[k] =      f11[k] * nl1tr[1] + f11[nv3[k]] * nl1tl[1] +
                      f11[nv7[k]] * nl1bl[1] + f11[nv4[k]] * nl1br[1];

            ff12[k] =      f12[k] * nl1tl[2] + f12[nv4[k]] * nl1bl[2] +
                      f12[nv1[k]] * nl1tr[2] + f12[nv8[k]] * nl1br[2];

            ff13[k] =      f13[k] * nl1bl[3] + f13[nv1[k]] * nl1br[3] +
                      f13[nv2[k]] * nl1tl[3] + f13[nv5[k]] * nl1tr[3];

            ff14[k] =      f14[k] * nl1br[4] + f14[nv3[k]] * nl1bl[4] +
                      f14[nv2[k]] * nl1tr[4] + f14[nv6[k]] * nl1tl[4];

            ff15[k] =      f15[k] * nl1tr[5] + f15[nv3[k]] * nl1tl[5] +
                      f15[nv7[k]] * nl1bl[5] + f15[nv4[k]] * nl1br[5];

            ff16[k] =      f16[k] * nl1tl[6] + f16[nv4[k]] * nl1bl[6] +
                      f16[nv1[k]] * nl1tr[6] + f16[nv8[k]] * nl1br[6];

            ff17[k] =      f17[k] * nl1bl[7] + f17[nv1[k]] * nl1br[7] +
                      f17[nv2[k]] * nl1tl[7] + f17[nv5[k]] * nl1tr[7];

            ff18[k] =      f18[k] * nl1br[8] + f18[nv3[k]] * nl1bl[8] +
                      f18[nv2[k]] * nl1tr[8] + f18[nv6[k]] * nl1tl[8];

            ff20[k] = f20[k];

            ff21[k] =      f21[k] * nl2tr[1] + f21[nv3[k]] * nl2tl[1] +
                      f21[nv7[k]] * nl2bl[1] + f21[nv4[k]] * nl2br[1];

            ff22[k] =      f22[k] * nl2tl[2] + f22[nv4[k]] * nl2bl[2] +
                      f22[nv1[k]] * nl2tr[2] + f22[nv8[k]] * nl2br[2];

            ff23[k] =      f23[k] * nl2bl[3] + f23[nv1[k]] * nl2br[3] +
                      f23[nv2[k]] * nl2tl[3] + f23[nv5[k]] * nl2tr[3];

            ff24[k] =      f24[k] * nl2br[4] + f24[nv3[k]] * nl2bl[4] +
                      f24[nv2[k]] * nl2tr[4] + f24[nv6[k]] * nl2tl[4];

            ff25[k] =      f25[k] * nl2tr[5] + f25[nv3[k]] * nl2tl[5] +
                      f25[nv7[k]] * nl2bl[5] + f25[nv4[k]] * nl2br[5];

            ff26[k] =      f26[k] * nl2tl[6] + f26[nv4[k]] * nl2bl[6] +
                      f26[nv1[k]] * nl2tr[6] + f26[nv8[k]] * nl2br[6];

            ff27[k] =      f27[k] * nl2bl[7] + f27[nv1[k]] * nl2br[7] +
```



```
                      f27[nv2[k]] * nl2tl[7] + f27[nv5[k]] * nl2tr[7];

            ff28[k] =      f28[k] * nl2br[8] + f28[nv3[k]] * nl2bl[8] +
                      f28[nv2[k]] * nl2tr[8] + f28[nv6[k]] * nl2tl[8];
            break;

        case 1:
            ff10[k] = f10[k];

            ff11[k] =      f11[k] * nlb1br[1] + f11[nv3[k]] * nlb1bl[1] +
                      f11[nv6[k]] * nlb1tl[1] + f11[nv2[k]] * nlb1tr[1];

            ff12[k] =      f12[k] * nlb1bl[2] + f12[nv1[k]] * nlb1br[2] +
                      f12[nv2[k]] * nlb1tl[2] + f12[nv5[k]] * nlb1tr[2];

            ff13[k] =      f13[k] * nl1bl[3] + f13[nv1[k]] * nl1br[3] +
                      f13[nv2[k]] * nl1tl[3] + f13[nv5[k]] * nl1tr[3];

            ff14[k] =      f14[k] * nl1br[4] + f14[nv3[k]] * nl1bl[4] +
                      f14[nv2[k]] * nl1tr[4] + f14[nv6[k]] * nl1tl[4];

            ff15[k] =      f15[k] * nlb1bl[5] + f15[nv1[k]] * nlb1br[5] +
                      f15[nv2[k]] * nlb1tl[5] + f15[nv5[k]] * nlb1tr[5];

            ff16[k] =      f16[k] * nlb1br[6] + f16[nv3[k]] * nlb1bl[6] +
                      f16[nv2[k]] * nlb1tr[6] + f16[nv6[k]] * nlb1tl[6];

            ff17[k] =      f17[k] * nl1bl[7] + f17[nv1[k]] * nl1br[7] +
                      f17[nv2[k]] * nl1tl[7] + f17[nv5[k]] * nl1tr[7];

            ff18[k] =      f18[k] * nl1br[8] + f18[nv3[k]] * nl1bl[8] +
                      f18[nv2[k]] * nl1tr[8] + f18[nv6[k]] * nl1tl[8];

            ff20[k] = f20[k];

            ff21[k] =      f21[k] * nlb2br[1] + f21[nv3[k]] * nlb2bl[1] +
                      f21[nv6[k]] * nlb2tl[1] + f21[nv2[k]] * nlb2tr[1];

            ff22[k] =      f22[k] * nlb2bl[2] + f22[nv1[k]] * nlb2br[2] +
                      f22[nv2[k]] * nlb2tl[2] + f22[nv5[k]] * nlb2tr[2];

            ff23[k] =      f23[k] * nl2bl[3] + f23[nv1[k]] * nl2br[3] +
                      f23[nv2[k]] * nl2tl[3] + f23[nv5[k]] * nl2tr[3];

            ff24[k] =      f24[k] * nl2br[4] + f24[nv3[k]] * nl2bl[4] +
                      f24[nv2[k]] * nl2tr[4] + f24[nv6[k]] * nl2tl[4];

            ff25[k] =      f25[k] * nlb2bl[5] + f25[nv1[k]] * nlb2br[5] +
                      f25[nv2[k]] * nlb2tl[5] + f25[nv5[k]] * nlb2tr[5];

            ff26[k] =      f26[k] * nlb2br[6] + f26[nv3[k]] * nlb2bl[6] +
                      f26[nv2[k]] * nlb2tr[6] + f26[nv6[k]] * nlb2tl[6];

            ff27[k] =      f27[k] * nl2bl[7] + f27[nv1[k]] * nl2br[7] +
                      f27[nv2[k]] * nl2tl[7] + f27[nv5[k]] * nl2tr[7];

            ff28[k] =      f28[k] * nl2br[8] + f28[nv3[k]] * nl2bl[8] +
                      f28[nv2[k]] * nl2tr[8] + f28[nv6[k]] * nl2tl[8];
            break;

        case 2:
            ff10[k] = f10[k];

            ff11[k] =      f11[k] * nl1tr[1] + f11[nv3[k]] * nl1tl[1] +
                      f11[nv7[k]] * nl1bl[1] + f11[nv4[k]] * nl1br[1];

            ff12[k] =      f12[k] * nl1tl[2] + f12[nv4[k]] * nl1bl[2] +
                      f12[nv1[k]] * nl1tr[2] + f12[nv8[k]] * nl1br[2];
```

```
    ff13[k] =       f13[k] * nlt1tl[3] + f13[nv1[k]] * nlt1tr[3] +
              f13[nv4[k]] * nlt1bl[3] + f13[nv8[k]] * nlt1br[3];

    ff14[k] =       f14[k] * nlt1tr[4] + f14[nv3[k]] * nlt1tl[4] +
              f14[nv4[k]] * nlt1br[4] + f14[nv7[k]] * nlt1bl[4];

    ff15[k] =       f15[k] * nl1tr[5] + f15[nv3[k]] * nl1tl[5] +
              f15[nv7[k]] * nl1bl[5] + f15[nv4[k]] * nl1br[5];

    ff16[k] =       f16[k] * nl1tl[6] + f16[nv4[k]] * nl1bl[6] +
              f16[nv1[k]] * nl1tr[6] + f16[nv8[k]] * nl1br[6];

    ff17[k] =       f17[k] * nlt1tr[7] + f17[nv3[k]] * nlt1tl[7] +
              f17[nv4[k]] * nlt1br[7] + f17[nv7[k]] * nlt1bl[7];

    ff18[k] =       f18[k] * nlt1tl[8] + f18[nv1[k]] * nlt1tr[8] +
              f18[nv4[k]] * nlt1bl[8] + f18[nv8[k]] * nlt1br[8];

    ff20[k] = f20[k];

    ff21[k] =       f21[k] * nl2tr[1] + f21[nv3[k]] * nl2tl[1] +
              f21[nv7[k]] * nl2bl[1] + f21[nv4[k]] * nl2br[1];

    ff22[k] =       f22[k] * nl2tl[2] + f22[nv4[k]] * nl2bl[2] +
              f22[nv1[k]] * nl2tr[2] + f22[nv8[k]] * nl2br[2];

    ff23[k] =       f23[k] * nlt2tl[3] + f23[nv1[k]] * nlt2tr[3] +
              f23[nv4[k]] * nlt2bl[3] + f23[nv8[k]] * nlt2br[3];

    ff24[k] =       f24[k] * nlt2tr[4] + f24[nv3[k]] * nlt2tl[4] +
              f24[nv4[k]] * nlt2br[4] + f24[nv7[k]] * nlt2bl[4];

    ff25[k] =       f25[k] * nl2tr[5] + f25[nv3[k]] * nl2tl[5] +
              f25[nv7[k]] * nl2bl[5] + f25[nv4[k]] * nl2br[5];

    ff26[k] =       f26[k] * nl2tl[6] + f26[nv4[k]] * nl2bl[6] +
              f26[nv1[k]] * nl2tr[6] + f26[nv8[k]] * nl2br[6];

    ff27[k] =       f27[k] * nlt2tr[7] + f27[nv3[k]] * nlt2tl[7] +
              f27[nv4[k]] * nlt2br[7] + f27[nv7[k]] * nlt2bl[7];

    ff28[k] =       f28[k] * nlt2tl[8] + f28[nv1[k]] * nlt2tr[8] +
              f28[nv4[k]] * nlt2bl[8] + f28[nv8[k]] * nlt2br[8];

    break;

  case 3:
    ff10[k] = f10[k];

    ff11[k] =       f11[k] * nl1ltl[1] + f11[nv1[k]] * nl1ltr[1] +
              f11[nv4[k]] * nl1lbl[1] + f11[nv8[k]] * nl1lbr[1];

    ff12[k] =       f12[k] * nl1tl[2] + f12[nv4[k]] * nl1bl[2] +
              f12[nv1[k]] * nl1tr[2] + f12[nv8[k]] * nl1br[2];

    ff13[k] =       f13[k] * nl1bl[3] + f13[nv1[k]] * nl1br[3] +
              f13[nv2[k]] * nl1tl[3] + f13[nv5[k]] * nl1tr[3];

    ff14[k] =       f14[k] * nl1lbl[4] + f14[nv1[k]] * nl1lbr[4] +
              f14[nv2[k]] * nl1ltl[4] + f14[nv5[k]] * nl1ltr[4];

    ff15[k] =       f15[k] * nl1lbl[5] + f15[nv1[k]] * nl1lbr[5] +
              f15[nv2[k]] * nl1ltl[5] + f15[nv5[k]] * nl1ltr[5];

    ff16[k] =       f16[k] * nl1tl[6] + f16[nv4[k]] * nl1bl[6] +
              f16[nv1[k]] * nl1tr[6] + f16[nv8[k]] * nl1br[6];

    ff17[k] =       f17[k] * nl1bl[7] + f17[nv1[k]] * nl1br[7] +
              f17[nv2[k]] * nl1tl[7] + f17[nv5[k]] * nl1tr[7];
```



```
    ff18[k] =       f18[k] * nl1ltl[8] + f18[nv1[k]] * nl1ltr[8] +
              f18[nv4[k]] * nl1lbl[8] + f18[nv8[k]] * nl1lbr[8];

    ff20[k] = f20[k];

    ff21[k] =       f21[k] * nl12tl[1] + f21[nv1[k]] * nl12tr[1] +
              f21[nv4[k]] * nl12bl[1] + f21[nv8[k]] * nl12br[1];

    ff22[k] =       f22[k] * nl2tl[2] + f22[nv4[k]] * nl2bl[2] +
              f22[nv1[k]] * nl2tr[2] + f22[nv8[k]] * nl2br[2];

    ff23[k] =       f23[k] * nl2bl[3] + f23[nv1[k]] * nl2br[3] +
              f23[nv2[k]] * nl2tl[3] + f23[nv5[k]] * nl2tr[3];

    ff24[k] =       f24[k] * nl12bl[4] + f24[nv1[k]] * nl12br[4] +
              f24[nv2[k]] * nl12tl[4] + f24[nv5[k]] * nl12tr[4];

    ff25[k] =       f25[k] * nl12bl[5] + f25[nv1[k]] * nl12br[5] +
              f25[nv2[k]] * nl12tl[5] + f25[nv5[k]] * nl12tr[5];

    ff26[k] =       f26[k] * nl2tl[6] + f26[nv4[k]] * nl2bl[6] +
              f26[nv1[k]] * nl2tr[6] + f26[nv8[k]] * nl2br[6];

    ff27[k] =       f27[k] * nl2bl[7] + f27[nv1[k]] * nl2br[7] +
              f27[nv2[k]] * nl2tl[7] + f27[nv5[k]] * nl2tr[7];

    ff28[k] =       f28[k] * nl12tl[8] + f28[nv1[k]] * nl12tr[8] +
              f28[nv4[k]] * nl12bl[8] + f28[nv8[k]] * nl12br[8];

    break;

  case 4:
    ff10[k] = f10[k];

    ff11[k] =       f11[k] * nl1tr[1] + f11[nv3[k]] * nl1tl[1] +
              f11[nv7[k]] * nl1bl[1] + f11[nv4[k]] * nl1br[1];

    ff12[k] =       f12[k] * nlr1tr[2] + f12[nv3[k]] * nlr1tl[2] +
              f12[nv4[k]] * nlr1br[2] + f12[nv7[k]] * nlr1bl[2];

    ff13[k] =       f13[k] * nlr1br[3] + f13[nv3[k]] * nlr1bl[3] +
              f13[nv2[k]] * nlr1tr[3] + f13[nv6[k]] * nlr1tl[3];

    ff14[k] =       f14[k] * nl1br[4] + f14[nv3[k]] * nl1bl[4] +
              f14[nv2[k]] * nl1tr[4] + f14[nv6[k]] * nl1tl[4];

    ff15[k] =       f15[k] * nl1tr[5] + f15[nv3[k]] * nl1tl[5] +
              f15[nv7[k]] * nl1bl[5] + f15[nv4[k]] * nl1br[5];

    ff16[k] =       f16[k] * nlr1br[6] + f16[nv3[k]] * nlr1bl[6] +
              f16[nv2[k]] * nlr1tr[6] + f16[nv6[k]] * nlr1tl[6];

    ff17[k] =       f17[k] * nlr1tr[7] + f17[nv3[k]] * nlr1tl[7] +
              f17[nv4[k]] * nlr1br[7] + f17[nv7[k]] * nlr1bl[7];

    ff18[k] =       f18[k] * nl1br[8] + f18[nv3[k]] * nl1bl[8] +
              f18[nv2[k]] * nl1tr[8] + f18[nv6[k]] * nl1tl[8];

    ff20[k] = f20[k];

    ff21[k] =       f21[k] * nl2tr[1] + f21[nv3[k]] * nl2tl[1] +
              f21[nv7[k]] * nl2bl[1] + f21[nv4[k]] * nl2br[1];

    ff22[k] =       f22[k] * nlr2tr[2] + f22[nv3[k]] * nlr2tl[2] +
              f22[nv4[k]] * nlr2br[2] + f22[nv7[k]] * nlr2bl[2];

    ff23[k] =       f23[k] * nlr2br[3] + f23[nv3[k]] * nlr2bl[3] +
              f23[nv2[k]] * nlr2tr[3] + f23[nv6[k]] * nlr2tl[3];
```

```
            ff24[k] =      f24[k] * nl2br[4] + f24[nv3[k]] * nl2bl[4] +
                     f24[nv2[k]] * nl2tr[4] + f24[nv6[k]] * nl2tl[4];

            ff25[k] =      f25[k] * nl2tr[5] + f25[nv3[k]] * nl2tl[5] +
                     f25[nv7[k]] * nl2bl[5] + f25[nv4[k]] * nl2br[5];

            ff26[k] =      f26[k] * nlr2br[6] + f26[nv3[k]] * nlr2bl[6] +
                     f26[nv2[k]] * nlr2tr[6] + f26[nv6[k]] * nlr2tl[6];

            ff27[k] =      f27[k] * nlr2tr[7] + f27[nv3[k]] * nlr2tl[7] +
                     f27[nv4[k]] * nlr2br[7] + f27[nv7[k]] * nlr2bl[7];

            ff28[k] =      f28[k] * nl2br[8] + f28[nv3[k]] * nl2bl[8] +
                     f28[nv2[k]] * nl2tr[8] + f28[nv6[k]] * nl2tl[8];
            break;
        }
    }
}
```

```
/********************************************************************\
*                                                                    *
*   wet9up.c                                                         *
*                                                                    *
\********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "wet9head.h"

double compute_upwind_sources_bulk(int sigma, int index, int k, int nv,
                            double nf10[], double nf11[], double nf12[],
                            double nf13[], double nf14[], double nf15[],
                            double nf16[], double nf17[], double nf18[],
                            double nf20[], double nf21[], double nf22[],
                            double nf23[], double nf24[], double nf25[],
                            double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, cfl, source;
  double gradx, grady;
  double fs1[9], fs2[9];

  switch(sigma)
    {
    case 1:
      cfl = cfl1;
      break;
    case 2:
      cfl = cfl2;
      break;
    }

  fs1[0] = nf10[k] - cfl * (nf10[k] - nf10[nv]);
  fs1[1] = nf11[k] - cfl * (nf11[k] - nf11[nv]);
  fs1[2] = nf12[k] - cfl * (nf12[k] - nf12[nv]);
  fs1[3] = nf13[k] - cfl * (nf13[k] - nf13[nv]);
  fs1[4] = nf14[k] - cfl * (nf14[k] - nf14[nv]);
  fs1[5] = nf15[k] - cfl * (nf15[k] - nf15[nv]);
  fs1[6] = nf16[k] - cfl * (nf16[k] - nf16[nv]);
  fs1[7] = nf17[k] - cfl * (nf17[k] - nf17[nv]);
  fs1[8] = nf18[k] - cfl * (nf18[k] - nf18[nv]);
  fs2[0] = nf20[k] - cfl * (nf20[k] - nf20[nv]);
  fs2[1] = nf21[k] - cfl * (nf21[k] - nf21[nv]);
  fs2[2] = nf22[k] - cfl * (nf22[k] - nf22[nv]);
  fs2[3] = nf23[k] - cfl * (nf23[k] - nf23[nv]);
  fs2[4] = nf24[k] - cfl * (nf24[k] - nf24[nv]);
  fs2[5] = nf25[k] - cfl * (nf25[k] - nf25[nv]);
  fs2[6] = nf26[k] - cfl * (nf26[k] - nf26[nv]);
  fs2[7] = nf27[k] - cfl * (nf27[k] - nf27[nv]);
  fs2[8] = nf28[k] - cfl * (nf28[k] - nf28[nv]);

  switch(boundary_mode[k])
        {
        case 0:
  dummy = mass1 * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+
                   fs1[6]+fs1[7]+fs1[8]) / tau1 +
          mass2 * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+
                   fs2[6]+fs2[7]+fs2[8]) / tau2;
  ux = (mass1 * (fs1[1]*ecx1[1]+fs1[2]*ecx1[2]+fs1[3]*ecx1[3]+fs1[4]*ecx1[4]+
                 fs1[5]*ecx1[5]+fs1[6]*ecx1[6]+fs1[7]*ecx1[7]+fs1[8]*ecx1[8])
        /tau1 +
        mass2 * (fs2[1]*ecx2[1]+fs2[2]*ecx2[2]+fs2[3]*ecx2[3]+fs2[4]*ecx2[4]+
                 fs2[5]*ecx2[5]+fs2[6]*ecx2[6]+fs2[7]*ecx2[7]+fs2[8]*ecx2[8])
        /tau2) / dummy;
  uy = (mass1 * (fs1[1]*ecy1[1]+fs1[2]*ecy1[2]+fs1[3]*ecy1[3]+fs1[4]*ecy1[4]+
                 fs1[5]*ecy1[5]+fs1[6]*ecy1[6]+fs1[7]*ecy1[7]+fs1[8]*ecy1[8])
        /tau1 +
```



```
        mass2 * (fs2[1]*ecy2[1]+fs2[2]*ecy2[2]+fs2[3]*ecy2[3]+fs2[4]*ecy2[4]+
                 fs2[5]*ecy2[5]+fs2[6]*ecy2[6]+fs2[7]*ecy2[7]+fs2[8]*ecy2[8])
        /tau2) / dummy;

  break;
        case 1:
          ux = uxwall_bot;
          uy = uywall_bot;
          break;
        case 2:
          ux = uxwall_top;
          uy = uywall_top;
          break;
        case 3:
          ux = uxwall_left;
          uy = uywall_left;
          break;
        case 4:
          ux = uxwall_right;
          uy = uywall_right;
          break;
        }
  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      /*
      printf("dummy=%25.20e\n",dummy);
      */
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      if(key_init < 3)
        {
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
        }
      else
        {
          gradx = gradn2x[k] - cfl * (gradn2x[k] - gradn2x[nv]);
          grady = gradn2y[k] - cfl * (gradn2y[k] - gradn2y[nv]);
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy +
              gforce * (gradx * (ecx1[index] - ux) +
                        grady * (ecy1[index] - uy))) * neq;
          /*
          source = - ctau1 * ( fs1[index] - neq ) + kforce * neq *
            (nloc2[k] - nloc2[nv]) *
            (cspeed1 - ecx[index] * ux - ecy[index] * uy);
            */
        }
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      if(key_init < 3)
        {
          source =  - ctau2 * ( fs2[index] - neq ) +
            ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
        }
      else
        {
          gradx = gradn1x[k] - cfl * (gradn1x[k] - gradn1x[nv]);
          grady = gradn1y[k] - cfl * (gradn1y[k] - gradn1y[nv]);
```

```
        source =  - ctau2 * ( fs2[index] - neq ) +
          ( ecprod2[index] - csforce_x * ux - csforce_y * uy +
            gforce * (gradx * (ecx2[index] - ux) +
                      grady * (ecy2[index] - uy))) * neq;
        /*
        source = - ctau2 * ( fs2[index] - neq ) + kforce * neq *
          (nloc1[k] - nloc1[nv]) *
          (cspeed2 - ecx[index] * ux - ecy[index] * uy);
          */
        /*
        printf("iter=%d  k=%d  gradx=%e  grady=%e  source=%e  gf=%e\n",
               iter,k,gradx,grady,source,
               gforce* (gradx * (ecx2[index]-ux) + grady * (ecy2[index]-uy)));
               */
      }
      break;
    }
  return source;
}

double compute_upwind_sources_boundary(int sigma, int index, int k, int nv,
                                double ux_boundary, double uy_boundary,
                                double nf10[], double nf11[], double nf12[],
                                double nf13[], double nf14[], double nf15[],
                                double nf16[], double nf17[], double nf18[],
                                double nf20[], double nf21[], double nf22[],
                                double nf23[], double nf24[], double nf25[],
                                double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, cfl, source;
  double gradx, grady;
  double fs1[9], fs2[9];

  switch(sigma)
    {
    case 1:
      cfl = cfl1;
      fs1[0] = nf10[k] - cfl * (nf10[nv] - nf10[k]);
      fs1[1] = nf11[k] - cfl * (nf11[nv] - nf11[k]);
      fs1[2] = nf12[k] - cfl * (nf12[nv] - nf12[k]);
      fs1[3] = nf13[k] - cfl * (nf13[nv] - nf13[k]);
      fs1[4] = nf14[k] - cfl * (nf14[nv] - nf14[k]);
      fs1[5] = nf15[k] - cfl * (nf15[nv] - nf15[k]);
      fs1[6] = nf16[k] - cfl * (nf16[nv] - nf16[k]);
      fs1[7] = nf17[k] - cfl * (nf17[nv] - nf17[k]);
      fs1[8] = nf18[k] - cfl * (nf18[nv] - nf18[k]);
      break;
    case 2:
      cfl = cfl2;
      fs2[0] = nf20[k] - cfl * (nf20[nv] - nf20[k]);
      fs2[1] = nf21[k] - cfl * (nf21[nv] - nf21[k]);
      fs2[2] = nf22[k] - cfl * (nf22[nv] - nf22[k]);
      fs2[3] = nf23[k] - cfl * (nf23[nv] - nf23[k]);
      fs2[4] = nf24[k] - cfl * (nf24[nv] - nf24[k]);
      fs2[5] = nf25[k] - cfl * (nf25[nv] - nf25[k]);
      fs2[6] = nf26[k] - cfl * (nf26[nv] - nf26[k]);
      fs2[7] = nf27[k] - cfl * (nf27[nv] - nf27[k]);
      fs2[8] = nf28[k] - cfl * (nf28[nv] - nf28[k]);
      break;
    }

  ux = ux_boundary - cfl * (uxloc[nv] - ux_boundary);
  uy = uy_boundary - cfl * (uyloc[nv] - uy_boundary);

  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
```



```
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                         fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      if(key_init < 3)
        {
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
        }
      else
        {
          gradx = gradn2x[k] - cfl * (gradn2x[k] - gradn2x[nv]);
          grady = gradn2y[k] - cfl * (gradn2y[k] - gradn2y[nv]);
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy -
              gforce * (gradx * ux + grady * uy)) * neq;
          /*
          source = - ctau1 * ( fs1[index] - neq ) + kforce * neq *
            (nloc2[k] - nloc2[nv]) *
            (cspeed1 - ecx[index] * ux - ecy[index] * uy);
            */
        }
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                         fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      if(key_init < 3)
        {
          source =  - ctau2 * ( fs2[index] - neq ) +
            ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
        }
      else
        {
          gradx = gradn1x[k] - cfl * (gradn1x[k] - gradn1x[nv]);
          grady = gradn1y[k] - cfl * (gradn1y[k] - gradn1y[nv]);
          source =  - ctau2 * ( fs2[index] - neq ) +
            ( ecprod2[index] - csforce_x * ux - csforce_y * uy -
              gforce * (gradx * ux + grady * uy)) * neq;
          /*
          source = - ctau2 * ( fs2[index] - neq ) + kforce * neq *
            (nloc1[k] - nloc1[nv]) *
            (cspeed2 - ecx[index] * ux - ecy[index] * uy);
            */
        }
      break;
    }
  return source;
}

double compute_upwind_sources_boundary_in(int sigma, int index, int k, int nv,
                                double ux_boundary, double uy_boundary,
                                double nf10[], double nf11[], double nf12[],
                                double nf13[], double nf14[], double nf15[],
                                double nf16[], double nf17[], double nf18[],
                                double nf20[], double nf21[], double nf22[],
                                double nf23[], double nf24[], double nf25[],
                                double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, cfl, source;
  double gradx, grady;
  double fs1[9], fs2[9];

  switch(sigma)
    {
    case 1:
      cfl = cfl1;
```

```
      fs1[0] = (nf10[k] - cfl * (nf10[nv] - nf10[k]));
      fs1[1] = (nf11[k] - cfl * (nf11[nv] - nf11[k]));
      fs1[2] = (nf12[k] - cfl * (nf12[nv] - nf12[k]));
      fs1[3] = (nf13[k] - cfl * (nf13[nv] - nf13[k]));
      fs1[4] = (nf14[k] - cfl * (nf14[nv] - nf14[k]));
      fs1[5] = (nf15[k] - cfl * (nf15[nv] - nf15[k]));
      fs1[6] = (nf16[k] - cfl * (nf16[nv] - nf16[k]));
      fs1[7] = (nf17[k] - cfl * (nf17[nv] - nf17[k]));
      fs1[8] = (nf18[k] - cfl * (nf18[nv] - nf18[k]));
      /*
      printf("k=%d  sigma=%d  index=%d  fs1=%e  nf21=%e  nf21[nv]=%e\n",
             k,sigma,index,fs1[1],nf11[k],nf11[nv]);
      */
      break;
    case 2:
      cfl = cfl2;
      fs2[0] = nf20[k] - cfl * (nf20[nv] - nf20[k]);
      fs2[1] = nf21[k] - cfl * (nf21[nv] - nf21[k]);
      fs2[2] = nf22[k] - cfl * (nf22[nv] - nf22[k]);
      fs2[3] = nf23[k] - cfl * (nf23[nv] - nf23[k]);
      fs2[4] = nf24[k] - cfl * (nf24[nv] - nf24[k]);
      fs2[5] = nf25[k] - cfl * (nf25[nv] - nf25[k]);
      fs2[6] = nf26[k] - cfl * (nf26[nv] - nf26[k]);
      fs2[7] = nf27[k] - cfl * (nf27[nv] - nf27[k]);
      fs2[8] = nf28[k] - cfl * (nf28[nv] - nf28[k]);
      /*
     printf("k=%d  sigma=%d  index=%d  fs2=%e  nf22=%e  nf22[nv]=%e\n",
             k,sigma,index,fs2[1],nf21[k],nf21[nv]);
      */
      break;
    }


  ux = ux_boundary - cfl * (uxloc[nv] - ux_boundary);
  uy = uy_boundary - cfl * (uyloc[nv] - uy_boundary);
  /*
  printf("iter=%d  k=%d  index=%d  ux=%e  ux_boundary=%e  uxloc[nv]=%e\n",
         iter,k,index,ux,ux_boundary,uxloc[nv]);
  */
  uu = ux*ux + uy*uy;
  switch(sigma)
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                 fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      if(key_init < 3)
        {
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
        }
      else
        {
          gradx = gradn2x[k] - cfl * (gradn2x[k] - gradn2x[nv]);
          grady = gradn2y[k] - cfl * (gradn2y[k] - gradn2y[nv]);
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy -
              gforce * (gradx * ux + grady * uy)) * neq;
          /*
          source = - ctau1 * ( fs1[index] - neq ) + kforce * neq *
            (nloc2[k] - nloc2[nv]) *
            (cspeed1 - ecx[index] * ux - ecy[index] * uy);
            */
        }
      /*
      if(neq < 0)
        {
```



```
          printf("negative neq sumfs1=%e  term=%e\n",
                 fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                 fs1[7]+fs1[8],
  1.000 + three * dummy + nine_over_two * dummy * dummy -
                 three_over_two * uu / cspeed12);
          printf("fs0=%e  nf10=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[0],nf10[k],nf10[nv],cfl);
          printf("fs1=%e  nf11=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[1],nf11[k],nf11[nv],cfl);
          printf("fs2=%e  nf12=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[2],nf12[k],nf12[nv],cfl);
          printf("fs3=%e  nf13=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[3],nf13[k],nf13[nv],cfl);
          printf("fs4=%e  nf14=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[4],nf14[k],nf14[nv],cfl);
          printf("fs5=%e  nf15=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[5],nf15[k],nf15[nv],cfl);
          printf("fs6=%e  nf16=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[6],nf16[k],nf16[nv],cfl);
          printf("fs7=%e  nf17=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[7],nf17[k],nf17[nv],cfl);
          printf("fs8=%e  nf18=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[8],nf18[k],nf18[nv],cfl);
}
        printf("iter=%d  k=%d  index=%d  fs1=%25.20e  neq=%25.20e  source=%25.20e  fs1-neq
=%25.20e  fs1+source=%25.20e  ux=%e  uy=%e  ecprod=%e\n",
               iter,k,index,fs1[index],neq,source,fs1[index]-neq,fs1[index]+source,ux,uy,e
cprod1[index]);
        */
        break;
      case 2:
        dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
        neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                   fs2[7]+fs2[8]) *
          (1.000 + three * dummy + nine_over_two * dummy * dummy -
          three_over_two * uu / cspeed22);
        if(key_init < 3)
          {
            source =  - ctau2 * ( fs2[index] - neq ) +
              ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
          }
        else
          {
            gradx = gradn1x[k] - cfl * (gradn1x[k] - gradn1x[nv]);
            grady = gradn1y[k] - cfl * (gradn1y[k] - gradn1y[nv]);
            source =  - ctau2 * ( fs2[index] - neq ) +
              ( ecprod2[index] - csforce_x * ux - csforce_y * uy -
                gforce * (gradx * ux + grady * uy)) * neq;
            /*
            source = - ctau2 * ( fs2[index] - neq ) + kforce * neq *
              (nloc1[k] - nloc1[nv]) *
              (cspeed2 - ecx[index] * ux - ecy[index] * uy);
              */
          }
        /*
        printf("iter=%d  k=%d  index=%d  fs2=%25.20e  neq=%25.20e  source=%25.20e  fs2-neq
=%25.20e  fs2+source=%25.20e  ux=%e  uy=%e  ecprod=%e\n",
               iter,k,index,fs2[index],neq,source,fs2[index]-neq,fs2[index]+source,ux,uy,e
cprod2[index]);

        if(neq < 0)
          {
          printf("negative neq sumfs2=%e  term=%e\n",
                 fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                 fs1[7]+fs1[8],
  1.000 + three * dummy + nine_over_two * dummy * dummy -
                 three_over_two * uu / cspeed12);
          printf("fs0=%e  nf10=%e  nf[nv]=%e  cfl=%e\n",
                 fs1[0],nf10[k],nf10[nv],cfl);
```

```
        printf("fs1=%e  nf11=%e  nf[nv]=%e  cfl=%e\n",
               fs1[1],nf11[k],nf11[nv],cfl);
        printf("fs2=%e  nf12=%e  nf[nv]=%e  cfl=%e\n",
               fs1[2],nf12[k],nf12[nv],cfl);
        printf("fs3=%e  nf13=%e  nf[nv]=%e  cfl=%e\n",
               fs1[3],nf13[k],nf13[nv],cfl);
        printf("fs4=%e  nf14=%e  nf[nv]=%e  cfl=%e\n",
               fs1[4],nf14[k],nf14[nv],cfl);
        printf("fs5=%e  nf15=%e  nf[nv]=%e  cfl=%e\n",
               fs1[5],nf15[k],nf15[nv],cfl);
        printf("fs6=%e  nf16=%e  nf[nv]=%e  cfl=%e\n",
               fs1[6],nf16[k],nf16[nv],cfl);
        printf("fs7=%e  nf17=%e  nf[nv]=%e  cfl=%e\n",
               fs1[7],nf17[k],nf17[nv],cfl);
        printf("fs8=%e  nf18=%e  nf[nv]=%e  cfl=%e\n",
               fs1[8],nf18[k],nf18[nv],cfl);
        }
      */
        break;
    }
  return source;
}

double compute_upwind_sources_boundary_out(int sigma, int index, int nv, int k,
                                     double ux_boundary, double uy_boundary,
                                     double nf10[], double nf11[], double nf12[],
                                     double nf13[], double nf14[], double nf15[],
                                     double nf16[], double nf17[], double nf18[],
                                     double nf20[], double nf21[], double nf22[],
                                     double nf23[], double nf24[], double nf25[],
                                     double nf26[], double nf27[], double nf28[])
{
  double dummy, ux, uy, uu, neq, cfl, source;
  double gradx, grady;
  double fs1[9], fs2[9];

  switch(sigma)
    {
    case 1:
      cfl = cfl1;
      fs1[0] = nf10[k] + cfl * (nf10[nv] - nf10[k]);
      fs1[1] = nf11[k] + cfl * (nf11[nv] - nf11[k]);
      fs1[2] = nf12[k] + cfl * (nf12[nv] - nf12[k]);
      fs1[3] = nf13[k] + cfl * (nf13[nv] - nf13[k]);
      fs1[4] = nf14[k] + cfl * (nf14[nv] - nf14[k]);
      fs1[5] = nf15[k] + cfl * (nf15[nv] - nf15[k]);
      fs1[6] = nf16[k] + cfl * (nf16[nv] - nf16[k]);
      fs1[7] = nf17[k] + cfl * (nf17[nv] - nf17[k]);
      fs1[8] = nf18[k] + cfl * (nf18[nv] - nf18[k]);
      break;
    case 2:
      cfl = cfl2;
      fs2[0] = nf20[k] + cfl * (nf20[nv] - nf20[k]);
      fs2[1] = nf21[k] + cfl * (nf21[nv] - nf21[k]);
      fs2[2] = nf22[k] + cfl * (nf22[nv] - nf22[k]);
      fs2[3] = nf23[k] + cfl * (nf23[nv] - nf23[k]);
      fs2[4] = nf24[k] + cfl * (nf24[nv] - nf24[k]);
      fs2[5] = nf25[k] + cfl * (nf25[nv] - nf25[k]);
      fs2[6] = nf26[k] + cfl * (nf26[nv] - nf26[k]);
      fs2[7] = nf27[k] + cfl * (nf27[nv] - nf27[k]);
      fs2[8] = nf28[k] + cfl * (nf28[nv] - nf28[k]);
      break;
    }

  ux = ux_boundary + cfl * (uxloc[nv] - ux_boundary);
  uy = uy_boundary + cfl * (uyloc[nv] - uy_boundary);

  uu = ux*ux + uy*uy;
  switch(sigma)
```



```
    {
    case 1:
      dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;
      neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+
                  fs1[7]+fs1[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed12);
      if(key_init < 3)
        {
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy) * neq;
        }
      else
        {
          gradx = gradn2x[k] - cfl * (gradn2x[k] - gradn2x[nv]);
          grady = gradn2y[k] - cfl * (gradn2y[k] - gradn2y[nv]);
          source =  - ctau1 * ( fs1[index] - neq ) +
            ( ecprod1[index] - csforce_x * ux - csforce_y * uy -
              gforce * (gradx * ux + grady * uy)) * neq;
          /*
          source = - ctau1 * ( fs1[index] - neq ) + kforce * neq *
            (nloc2[k] - nloc2[nv]) *
            (cspeed1 - ecx[index] * ux - ecy[index] * uy);
            */
        }
      break;
    case 2:
      dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22;
      neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+
                  fs2[7]+fs2[8]) *
        (1.000 + three * dummy + nine_over_two * dummy * dummy -
         three_over_two * uu / cspeed22);
      if(key_init < 3)
        {
          source =  - ctau2 * ( fs2[index] - neq ) +
            ( ecprod2[index] - csforce_x * ux - csforce_y * uy) * neq;
        }
      else
        {
          gradx = gradn1x[k] - cfl * (gradn1x[k] - gradn1x[nv]);
          grady = gradn1y[k] - cfl * (gradn1y[k] - gradn1y[nv]);
          source =  - ctau2 * ( fs2[index] - neq ) +
            ( ecprod2[index] - csforce_x * ux - csforce_y * uy -
              gforce * (gradx * ux + grady * uy)) * neq;
          /*
          source = - ctau2 * ( fs2[index] - neq ) + kforce * neq *
            (nloc1[k] - nloc1[nv]) *
            (cspeed2 - ecx[index] * ux - ecy[index] * uy);
            */
        }
        break;
    }
  return source;
}

void compute_upwind_sources(double nf10[], double nf11[], double nf12[],
                      double nf13[], double nf14[], double nf15[],
                      double nf16[], double nf17[], double nf18[],
                      double nf20[], double nf21[], double nf22[],
                      double nf23[], double nf24[], double nf25[],
                      double nf26[], double nf27[], double nf28[])
{
  int k;

  for(k=0; k<nnodes_all; k++)
    {
      sf10[k] = nf10[k];
      sf20[k] = nf20[k];
      source10[k] = - ctau1 * (nf10[k] - neq10[k]) -
```

```
      ( csforce_x * uxloc[k] + csforce_y * uyloc[k] ) * neq10[k];
    source20[k] = - ctau2 * (nf20[k] - neq20[k]) -
      ( csforce_x * uxloc[k] + csforce_y * uyloc[k] ) * neq20[k];

    switch(boundary_mode[k])
      {
      case 0:
        sf11[k] = nf11[k] - cf11 * (nf11[k] - nf11[nv3[k]]);
        source11[k] = compute_upwind_sources_bulk(1, 1, k, nv3[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf21[k] = nf21[k] - cf12 * (nf21[k] - nf21[nv3[k]]);
        source21[k] = compute_upwind_sources_bulk(2, 1, k, nv3[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf12[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]);
        source12[k] = compute_upwind_sources_bulk(1, 2, k, nv4[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf22[k] = nf22[k] - cf12 * (nf22[k] - nf22[nv4[k]]);
        source22[k] = compute_upwind_sources_bulk(2, 2, k, nv4[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf13[k] = nf13[k] - cf11 * (nf13[k] - nf13[nv1[k]]);
        source13[k] = compute_upwind_sources_bulk(1, 3, k, nv1[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf23[k] = nf23[k] - cf12 * (nf23[k] - nf23[nv1[k]]);
        source23[k] = compute_upwind_sources_bulk(2, 3, k, nv1[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]);
        source14[k] = compute_upwind_sources_bulk(1, 4, k, nv2[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf24[k] = nf24[k] - cf12 * (nf24[k] - nf24[nv2[k]]);
        source24[k] = compute_upwind_sources_bulk(2, 4, k, nv2[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]);
        source15[k] = compute_upwind_sources_bulk(1, 5, k, nv7[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf25[k] = nf25[k] - cf12 * (nf25[k] - nf25[nv7[k]]);
        source25[k] = compute_upwind_sources_bulk(2, 5, k, nv7[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf16[k] = nf16[k] - cf11 * (nf16[k] - nf16[nv8[k]]);
        source16[k] = compute_upwind_sources_bulk(1, 6, k, nv8[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf26[k] = nf26[k] - cf12 * (nf26[k] - nf26[nv8[k]]);
        source26[k] = compute_upwind_sources_bulk(2, 6, k, nv8[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]);
        source17[k] = compute_upwind_sources_bulk(1, 7, k, nv5[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]);
        source27[k] = compute_upwind_sources_bulk(2, 7, k, nv5[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
```



```
        sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]);
        source18[k] = compute_upwind_sources_bulk(1, 8, k, nv6[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf28[k] = nf28[k] - cf12 * (nf28[k] - nf28[nv6[k]]);
        source28[k] = compute_upwind_sources_bulk(2, 8, k, nv6[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        break;
      case 1:  /*  bottom wall  */
        sf11[k] = nf11[k] - cf11 * (nf11[k] - nf11[nv3[k]]);
        source11[k] = compute_upwind_sources_bulk(1, 1, k, nv3[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf21[k] = nf21[k] - cf12 * (nf21[k] - nf21[nv3[k]]);
        source21[k] = compute_upwind_sources_bulk(2, 1, k, nv3[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf12[k] = nf12[k] - cf11 * (nf12[nv2[k]] - nf12[k]);
        source12[k] = compute_upwind_sources_boundary_in(1, 2, k, nv2[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf22[k] = nf22[k] - cf12 * (nf22[nv2[k]] - nf22[k]);
        source22[k] = compute_upwind_sources_boundary_in(2, 2, k, nv2[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf13[k] = nf13[k] - cf11 * (nf13[k] - nf13[nv1[k]]);
        source13[k] = compute_upwind_sources_bulk(1, 3, k, nv1[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf23[k] = nf23[k] - cf12 * (nf23[k] - nf23[nv1[k]]);
        source23[k] = compute_upwind_sources_bulk(2, 3, k, nv1[k],
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]);
        source14[k] = compute_upwind_sources_boundary_out(1, 4, nv2[k], k,
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf24[k] = nf24[k] - cf12 * (nf24[k] - nf24[nv2[k]]);
        source24[k] = compute_upwind_sources_boundary_out(2, 4, nv2[k], k,
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf15[k] = nf15[k] - cf11 * (nf15[nv5[k]] - nf15[k]);
        source15[k] = compute_upwind_sources_boundary_in(1, 5, k, nv5[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf25[k] = nf25[k] - cf12 * (nf25[nv5[k]] - nf25[k]);
        source25[k] = compute_upwind_sources_boundary_in(2, 5, k, nv5[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

        sf16[k] = nf16[k] - cf11 * (nf16[nv6[k]] - nf16[k]);
        source16[k] = compute_upwind_sources_boundary_in(1, 6, k, nv6[k],
                      uxwall_bot, uywall_bot,
                      nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                      nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
        sf26[k] = nf26[k] - cf12 * (nf26[nv6[k]] - nf26[k]);
        source26[k] = compute_upwind_sources_boundary_in(2, 6, k, nv6[k],
```

```
                    uxwall_bot, uywall_bot,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]);
   source17[k] = compute_upwind_sources_boundary_out(1, 7, nv5[k], k,
                    uxwall_bot, uywall_bot,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]);
   source27[k] = compute_upwind_sources_boundary_out(2, 7, nv5[k], k,
                    uxwall_bot, uywall_bot,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]);
   source18[k] = compute_upwind_sources_boundary_out(1, 8, nv6[k], k,
                    uxwall_bot, uywall_bot,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf28[k] = nf28[k] - cf12 * (nf28[k] - nf28[nv6[k]]);
   source28[k] = compute_upwind_sources_boundary_out(2, 8, nv6[k], k,
                    uxwall_bot, uywall_bot,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   break;
  case 2:  /*  top wall  */
   sf11[k] = nf11[k] - cf11 * (nf11[k] - nf11[nv3[k]]);
   source11[k] = compute_upwind_sources_bulk(1, 1, k, nv3[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf21[k] = nf21[k] - cf12 * (nf21[k] - nf21[nv3[k]]);
   source21[k] = compute_upwind_sources_bulk(2, 1, k, nv3[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf12[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]);
   source12[k] = compute_upwind_sources_boundary_out(1, 2, nv4[k], k,
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf22[k] = nf22[k] - cf12 * (nf22[k] - nf22[nv4[k]]);
   source22[k] = compute_upwind_sources_boundary_out(2, 2, nv4[k], k,
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf13[k] = nf13[k] - cf11 * (nf13[k] - nf13[nv1[k]]);
   source13[k] = compute_upwind_sources_bulk(1, 3, k, nv1[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf23[k] = nf23[k] - cf12 * (nf23[k] - nf23[nv1[k]]);
   source23[k] = compute_upwind_sources_bulk(2, 3, k, nv1[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf14[k] = nf14[k] - cf11 * (nf14[nv4[k]] - nf14[k]);
   source14[k] = compute_upwind_sources_boundary_in(1, 4, k, nv4[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf24[k] = nf24[k] - cf12 * (nf24[nv4[k]] - nf24[k]);
   source24[k] = compute_upwind_sources_boundary_in(2, 4, k, nv4[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]);
   source15[k] = compute_upwind_sources_boundary_out(1, 5, nv7[k], k,
```



```
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf25[k] = nf25[k] - cf12 * (nf25[k] - nf25[nv7[k]]);
   source25[k] = compute_upwind_sources_boundary_out(2, 5, nv7[k], k,
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf16[k] = nf16[k] - cf11 * (nf16[k] - nf16[nv8[k]]);
   source16[k] = compute_upwind_sources_boundary_out(1, 6, nv8[k], k,
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf26[k] = nf26[k] - cf12 * (nf26[k] - nf26[nv8[k]]);
   source26[k] = compute_upwind_sources_boundary_out(2, 6, nv8[k], k,
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf17[k] = nf17[k] - cf11 * (nf17[nv7[k]] - nf17[k]);
   source17[k] = compute_upwind_sources_boundary_in(1, 7, k, nv7[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf27[k] = nf27[k] - cf12 * (nf27[nv7[k]] - nf27[k]);
   source27[k] = compute_upwind_sources_boundary_in(2, 7, k, nv7[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf18[k] = nf18[k] - cf11 * (nf18[nv8[k]] - nf18[k]);
   source18[k] = compute_upwind_sources_boundary_in(1, 8, k, nv8[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf28[k] = nf28[k] - cf12 * (nf28[nv8[k]] - nf28[k]);
   source28[k] = compute_upwind_sources_boundary_in(2, 8, k, nv8[k],
                    uxwall_top, uywall_top,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   break;
  case 3:
   sf11[k] = nf11[k] - cf11 * (nf11[nv1[k]] - nf11[k]);
   source11[k] = compute_upwind_sources_boundary_in(1, 1, k, nv1[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf21[k] = nf21[k] - cf12 * (nf21[nv1[k]] - nf21[k]);
   source21[k] = compute_upwind_sources_boundary_in(2, 1, k, nv1[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf12[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]);
   source12[k] = compute_upwind_sources_bulk(1, 2, k, nv4[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf22[k] = nf22[k] - cf12 * (nf22[k] - nf22[nv4[k]]);
   source22[k] = compute_upwind_sources_bulk(2, 2, k, nv4[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

   sf13[k] = nf13[k] - cf11 * (nf13[k] - nf13[nv1[k]]);
   source13[k] = compute_upwind_sources_boundary_out(1, 3, nv1[k], k,
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
   sf23[k] = nf23[k] - cf12 * (nf23[k] - nf23[nv1[k]]);
```

```
    source23[k] = compute_upwind_sources_boundary_out(2, 3, nv1[k], k,
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]);
    source14[k] = compute_upwind_sources_bulk(1, 4, k, nv2[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf24[k] = nf24[k] - cf12 * (nf24[k] - nf24[nv2[k]]);
    source24[k] = compute_upwind_sources_bulk(2, 4, k, nv2[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf15[k] = nf15[k] - cf11 * (nf15[nv5[k]] - nf15[k]);
    source15[k] = compute_upwind_sources_boundary_in(1, 5, k, nv5[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf25[k] = nf25[k] - cf12 * (nf25[nv5[k]] - nf25[k]);
    source25[k] = compute_upwind_sources_boundary_in(2, 5, k, nv5[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf16[k] = nf16[k] - cf11 * (nf16[k] - nf16[nv8[k]]);
    source16[k] = compute_upwind_sources_boundary_out(1, 6, nv8[k], k,
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf26[k] = nf26[k] - cf12 * (nf26[k] - nf26[nv8[k]]);
    source26[k] = compute_upwind_sources_boundary_out(2, 6, nv8[k], k,
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]);
    source17[k] = compute_upwind_sources_boundary_out(1, 7, nv5[k], k,
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]);
    source27[k] = compute_upwind_sources_boundary_out(2, 7, nv5[k], k,
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf18[k] = nf18[k] - cf11 * (nf18[nv8[k]] - nf18[k]);
    source18[k] = compute_upwind_sources_boundary_in(1, 8, k, nv8[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf28[k] = nf28[k] - cf12 * (nf28[nv8[k]] - nf28[k]);
    source28[k] = compute_upwind_sources_boundary_in(2, 8, k, nv8[k],
                    uxwall_left, uywall_left,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    break;
  case 4:
    sf11[k] = nf11[k] - cf11 * (nf11[k] - nf11[nv3[k]]);
    source11[k] = compute_upwind_sources_boundary_out(1, 1, nv3[k], k,
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf21[k] = nf21[k] - cf12 * (nf21[k] - nf21[nv3[k]]);
    source21[k] = compute_upwind_sources_boundary_out(2, 1, nv3[k], k,
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
```



```
    sf12[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]);
    source12[k] = compute_upwind_sources_bulk(1, 2, k, nv4[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf22[k] = nf22[k] - cf12 * (nf22[k] - nf22[nv4[k]]);
    source22[k] = compute_upwind_sources_bulk(2, 2, k, nv4[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf13[k] = nf13[k] - cf11 * (nf13[nv3[k]] - nf13[k]);
    source13[k] = compute_upwind_sources_boundary_in(1, 3, k, nv3[k],
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf23[k] = nf23[k] - cf12 * (nf23[nv3[k]] - nf23[k]);
    source23[k] = compute_upwind_sources_boundary_in(2, 3, k, nv3[k],
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]);
    source14[k] = compute_upwind_sources_bulk(1, 4, k, nv2[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf24[k] = nf24[k] - cf12 * (nf24[k] - nf24[nv2[k]]);
    source24[k] = compute_upwind_sources_bulk(2, 4, k, nv2[k],
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]);
    source15[k] = compute_upwind_sources_boundary_out(1, 5, nv7[k], k,
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf25[k] = nf25[k] - cf12 * (nf25[k] - nf25[nv7[k]]);
    source25[k] = compute_upwind_sources_boundary_out(2, 5, nv7[k], k,
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf16[k] = nf16[k] - cf11 * (nf16[nv6[k]] - nf16[k]);
    source16[k] = compute_upwind_sources_boundary_in(1, 6, k, nv6[k],
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf26[k] = nf26[k] - cf12 * (nf26[nv6[k]] - nf26[k]);
    source26[k] = compute_upwind_sources_boundary_in(2, 6, k, nv6[k],
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf17[k] = nf17[k] - cf11 * (nf17[nv7[k]] - nf17[k]);
    source17[k] = compute_upwind_sources_boundary_in(1, 7, k, nv7[k],
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf27[k] = nf27[k] - cf12 * (nf27[nv7[k]] - nf27[k]);
    source27[k] = compute_upwind_sources_boundary_in(2, 7, k, nv7[k],
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);

    sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]);
    source18[k] = compute_upwind_sources_boundary_out(1, 8, nv6[k], k,
                    uxwall_right, uywall_right,
                    nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                    nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
    sf28[k] = nf28[k] - cf12 * (nf28[k] - nf28[nv6[k]]);
```

```
          source28[k] = compute_upwind_sources_boundary_out(2, 8, nv6[k], k,
                          uxwall_right, uywall_right,
                          nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,
                          nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
          break;
        }
    }
}

void upwind_cfl(double mass, double tau, double cspeed, double cfl,
                double nf0[], double nf1[], double nf2[],
                double nf3[], double nf4[], double nf5[],
                double nf6[], double nf7[], double nf8[],
                double nff0[], double nff1[], double nff2[],
                double nff3[], double nff4[], double nff5[],
                double nff6[], double nff7[], double nff8[],
                double sf0[], double sf1[], double sf2[],
                double sf3[], double sf4[], double sf5[],
                double sf6[], double sf7[], double sf8[],
                double neq0[], double neq1[], double neq2[],
                double neq3[], double neq4[], double neq5[],
                double neq6[], double neq7[], double neq8[],
                double source0[], double source1[], double source2[],
                double source3[], double source4[], double source5[],
                double source6[], double source7[], double source8[])
{
  int k;
  double ctau, prodscal;
  ctau = delta_t / tau;

  for(k=0; k<nnodes_all; k++)
    {
     prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k];
     nff0[k] = sf0[k] + source0[k];
     nff1[k] = sf1[k] + (source1[k] - ctau * (nf1[k] - neq1[k]) +
                         cfl * ( ecprod1[1] - prodscal) * neq1[k]) / 2.000;
     nff2[k] = sf2[k] + (source2[k] - ctau * (nf2[k] - neq2[k]) +
                         cfl * ( ecprod1[2] - prodscal) * neq2[k]) / 2.000;
     nff3[k] = sf3[k] + (source3[k] - ctau * (nf3[k] - neq3[k]) +
                         cfl * ( ecprod1[3] - prodscal) * neq3[k]) / 2.000;
     nff4[k] = sf4[k] + (source4[k] - ctau * (nf4[k] - neq4[k]) +
                         cfl * ( ecprod1[4] - prodscal) * neq4[k]) / 2.000;
     nff5[k] = sf5[k] + (source5[k] - ctau * (nf5[k] - neq5[k]) +
                         cfl * ( ecprod1[5] - prodscal) * neq5[k]) / 2.000;
     nff6[k] = sf6[k] + (source6[k] - ctau * (nf6[k] - neq6[k]) +
                         cfl * ( ecprod1[6] - prodscal) * neq6[k]) / 2.000;
     nff7[k] = sf7[k] + (source7[k] - ctau * (nf7[k] - neq7[k]) +
                         cfl * ( ecprod1[7] - prodscal) * neq7[k]) / 2.000;
     nff8[k] = sf8[k] + (source8[k] - ctau * (nf8[k] - neq8[k]) +
                         cfl * ( ecprod1[8] - prodscal) * neq8[k]) / 2.000;
     /*
     nff2[k] = sf2[k] + source2[k];
     nff3[k] = sf3[k] + source3[k];
     nff4[k] = sf4[k] + source4[k];
     nff5[k] = sf5[k] + source5[k];
     nff6[k] = sf6[k] + source6[k];
     nff7[k] = sf7[k] + source7[k];
     nff8[k] = sf8[k] + source8[k];
     */
    }
}
```

```
/***********************************************************************\
*                                                                       *
*  wet9main.c                                                           *
*                                                                       *
\***********************************************************************/

#define MAIN_HEADER

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "wet9head.h"

void getavec_square(void);
void wet9_input(void);
void wet9_output(void);
void build_names(int);
void test_ntot(void);
void test_distribution_functions(double n0[], double n1[], double n2[],
                                 double n3[], double n4[], double n5[],
                                 double n6[], double n7[], double n8[]);
void xv_new(double n0[], double n1[], double n2[], double n3[],
            double n4[], double n5[], double n6[], double n7[],
            double n8[], char filename[], double maxval);
void xv(double n0[], double n1[], double n2[], double n3[],
        double n4[], double n5[], double n6[], double n7[],
        double n8[], char filename[]);
void init_ecx_nine(void);
void init_arrays_nine_square(void);
void init_arrays_nine_square_aux(void);
void wet9_channel_profile(void);
void wet9_drop_profile(void);


void compute_local_speeds(double nf10[], double nf11[], double nf12[],
                          double nf13[], double nf14[], double nf15[],
                          double nf16[], double nf17[], double nf18[],
                          double nf20[], double nf21[], double nf22[],
                          double nf23[], double nf24[], double nf25[],
                          double nf26[], double nf27[], double nf28[])
{
  int k;
  double dummy, uxloc1, uyloc1, uxloc2, uyloc2;

  for(k=0; k<nnodes_all; k++)
    {
      nloc1[k] = nf10[k]+nf11[k]+nf12[k]+nf13[k]+nf14[k]+nf15[k]+nf16[k]+
                 nf17[k]+nf18[k];
      nloc2[k] = nf20[k]+nf21[k]+nf22[k]+nf23[k]+nf24[k]+nf25[k]+nf26[k]+
                 nf27[k]+nf28[k];
      colorfield[k] = nloc1[k] - nloc2[k];
      switch(boundary_mode[k])
        {
        case 0:
        case 11:
        case 22:
          uxloc1 = (nf11[k]*ecx1[1]+nf12[k]*ecx1[2]+nf13[k]*ecx1[3]+
                    nf14[k]*ecx1[4]+nf15[k]*ecx1[5]+nf16[k]*ecx1[6]+
                    nf17[k]*ecx1[7]+nf18[k]*ecx1[8]);
          uyloc1 = (nf11[k]*ecy1[1]+nf12[k]*ecy1[2]+nf13[k]*ecy1[3]+
                    nf14[k]*ecy1[4]+nf15[k]*ecy1[5]+nf16[k]*ecy1[6]+
                    nf17[k]*ecy1[7]+nf18[k]*ecy1[8]);

          uxloc2 = (nf21[k]*ecx2[1]+nf22[k]*ecx2[2]+nf23[k]*ecx2[3]+
                    nf24[k]*ecx2[4]+nf25[k]*ecx2[5]+nf26[k]*ecx2[6]+
                    nf27[k]*ecx2[7]+nf28[k]*ecx2[8]);
          uyloc2 = (nf21[k]*ecy2[1]+nf22[k]*ecy2[2]+nf23[k]*ecy2[3]+
                    nf24[k]*ecy2[4]+nf25[k]*ecy2[5]+nf26[k]*ecy2[6]+
```



```
                    nf27[k]*ecy2[7]+nf28[k]*ecy2[8]);
          dummy = (mass1*nloc1[k])/tau1 + (mass2*nloc2[k])/tau2;
          if(dummy)
            {
              uxloc[k] = ((mass1 * uxloc1) / tau1 +
                          (mass2 * uxloc2) / tau2) / dummy;
              uyloc[k] = ((mass1 * uyloc1) / tau1 +
                          (mass2 * uyloc2) / tau2) / dummy;
            }
          else
            {
              uxloc[k] = 0.0000;
              uyloc[k] = 0.0000;
            }
          /*
          printf("k=%d  uxloc=%lf  uyloc=%lf  mass1=%lf  mass2=%lf  tau1=%e   tau2=%e  n
1=%e  n2=%e  cspeed2=%e  cspeed22=%e\n",
                 k,uxloc[k],uyloc[k],mass1,mass2,tau1,tau2,nloc1[k],nloc2[k],
                 cspeed2,cspeed22);
          */
          break;
        case 1:
          uxloc[k] = uxwall_bot;
          uyloc[k] = uywall_bot;
          break;
        case 2:
          uxloc[k] = uxwall_top;
          uyloc[k] = uywall_top;
          break;
        case 3:
          uxloc[k] = uxwall_left;
          uyloc[k] = uywall_left;
          break;
        case 4:
          uxloc[k] = uxwall_right;
          uyloc[k] = uywall_right;
          break;
        }
    }
}

void compute_equilibrium_distributions(void)
{
  int k;
  double dummy, uu, uu1, uu2;

  for(k=0; k<nnodes_all; k++)
    {
      uu = uxloc[k]*uxloc[k]+uyloc[k]*uyloc[k];
      uu1 = three_over_two * uu / cspeed12;
      uu2 = three_over_two * uu / cspeed22;

      neq10[k] = w0*nloc1[k] * (1.000 - uu1);

      dummy = (ecx1[1]*uxloc[k] + ecy1[1]*uyloc[k]) / cspeed12;
      neq11[k] = w1*nloc1[k] * (1.000 + three * dummy +
                             nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[2]*uxloc[k] + ecy1[2]*uyloc[k]) / cspeed12;
      neq12[k] = w1*nloc1[k] * (1.000 + three * dummy +
                             nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[3]*uxloc[k] + ecy1[3]*uyloc[k]) / cspeed12;
      neq13[k] = w1*nloc1[k] * (1.000 + three * dummy +
                             nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[4]*uxloc[k] + ecy1[4]*uyloc[k]) / cspeed12;
      neq14[k] = w1*nloc1[k] * (1.000 + three * dummy +
                             nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[5]*uxloc[k] + ecy1[5]*uyloc[k]) / cspeed12;
      neq15[k] = w2*nloc1[k] * (1.000 + three * dummy +
                             nine_over_two * dummy * dummy - uu1);
```

```
      dummy = (ecx1[6]*uxloc[k] + ecy1[6]*uyloc[k]) / cspeed12;
      neq16[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[7]*uxloc[k] + ecy1[7]*uyloc[k]) / cspeed12;
      neq17[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[8]*uxloc[k] + ecy1[8]*uyloc[k]) / cspeed12;
      neq18[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);

      neq20[k] = w0*nloc2[k] * (1.000 - uu2);

      dummy = (ecx2[1]*uxloc[k] + ecy2[1]*uyloc[k]) / cspeed22;
      neq21[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[2]*uxloc[k] + ecy2[2]*uyloc[k]) / cspeed22;
      neq22[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[3]*uxloc[k] + ecy2[3]*uyloc[k]) / cspeed22;
      neq23[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[4]*uxloc[k] + ecy2[4]*uyloc[k]) / cspeed22;
      neq24[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[5]*uxloc[k] + ecy2[5]*uyloc[k]) / cspeed22;
      neq25[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[6]*uxloc[k] + ecy2[6]*uyloc[k]) / cspeed22;
      neq26[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[7]*uxloc[k] + ecy2[7]*uyloc[k]) / cspeed22;
      neq27[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[8]*uxloc[k] + ecy2[8]*uyloc[k]) / cspeed22;
      neq28[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      /*
      printf("n1=%lf  neq1=%lf  n2=%lf  neq2=%lf\n",nloc1[k],
             neq10[k]+neq11[k]+neq12[k]+neq13[k]+neq14[k]+neq15[k]+
             neq16[k]+neq17[k]+neq18[k],nloc2[k],
             neq20[k]+neq21[k]+neq22[k]+neq23[k]+neq24[k]+neq25[k]+
             neq26[k]+neq27[k]+neq28[k]);
      */
    }
}

void compute_local_gradients(void)
{
  int k;
  double dummy = 3.000 / delta_x;

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:
          gradn1x[k] = (nloc1[nv1[k]] - nloc1[nv3[k]]) / (2.000 * delta_x);
          gradn2x[k] = (nloc2[nv1[k]] - nloc2[nv3[k]]) / (2.000 * delta_x);
          gradn1y[k] = (nloc1[nv2[k]] - nloc1[nv4[k]]) / (2.000 * delta_x);
          gradn2y[k] = (nloc2[nv2[k]] - nloc2[nv4[k]]) / (2.000 * delta_x);

          gradcolor_x[k] = (colorfield[nv1[k]] - colorfield[nv3[k]]) /
            (2.000 * delta_x);
          gradcolor_y[k] = (colorfield[nv2[k]] - colorfield[nv4[k]]) /
            (2.000 * delta_x);

          /*
          gradn1x[k] = dummy *
                       ( w1 * (
```



```
                         ecx[1]*nloc1[nv1[k]] + ecx[2]*nloc1[nv2[k]] +
                         ecx[3]*nloc1[nv3[k]] + ecx[4]*nloc1[nv4[k]]) +
                         w2 * (
                         ecx[5]*nloc1[nv5[k]] + ecx[6]*nloc1[nv6[k]] +
                         ecx[7]*nloc1[nv7[k]] + ecx[8]*nloc1[nv8[k]]));
          gradn2x[k] = dummy *
                       ( w1 * (
                         ecx[1]*nloc2[nv1[k]] + ecx[2]*nloc2[nv2[k]] +
                         ecx[3]*nloc2[nv3[k]] + ecx[4]*nloc2[nv4[k]]) +
                         w2 * (
                         ecx[5]*nloc2[nv5[k]] + ecx[6]*nloc2[nv6[k]] +
                         ecx[7]*nloc2[nv7[k]] + ecx[8]*nloc2[nv8[k]]));
          gradn1y[k] = dummy *
                       ( w1 * (
                         ecy[1]*nloc1[nv1[k]] + ecy[2]*nloc1[nv2[k]] +
                         ecy[3]*nloc1[nv3[k]] + ecy[4]*nloc1[nv4[k]]) +
                         w2 * (
                         ecy[5]*nloc1[nv5[k]] + ecy[6]*nloc1[nv6[k]] +
                         ecy[7]*nloc1[nv7[k]] + ecy[8]*nloc1[nv8[k]]));
          gradn2y[k] = dummy *
                       ( w1 * (
                         ecy[1]*nloc2[nv1[k]] + ecy[2]*nloc2[nv2[k]] +
                         ecy[3]*nloc2[nv3[k]] + ecy[4]*nloc2[nv4[k]]) +
                         w2 * (
                         ecy[5]*nloc2[nv5[k]] + ecy[6]*nloc2[nv6[k]] +
                         ecy[7]*nloc2[nv7[k]] + ecy[8]*nloc2[nv8[k]]));
          */
        break;
        case 1:
          break;
        case 2:
          break;
        }
    }
}


void first_upwind(double mass, double tau, double cspeed,
                  double nf0[], double nf1[], double nf2[],
                  double nf3[], double nf4[], double nf5[],
                  double nf6[], double nf7[], double nf8[],
                  double nff0[], double nff1[], double nff2[],
                  double nff3[], double nff4[], double nff5[],
                  double nff6[], double nff7[], double nff8[],
                  double neq0[], double neq1[], double neq2[],
                  double neq3[], double neq4[], double neq5[],
                  double neq6[], double neq7[], double neq8[],
                  double ecprod[], double sforce,
                  double gradx[], double grady[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxs2, cgradys2, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal, color_x, color_y, colormod, nn;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_x;
  cgradys2 = cspeed * sqrt((double) 2) * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
             (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));


  for(k=0; k<nnodes_all; k++)
```

```
      {
       prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k];
       switch(boundary_mode[k])
         {
         case 0:   /*  bulk  */
           nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
             (ecprod[0] - prodscal) * neq0[k];
           nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
             - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]) +
             (ecprod[1] - prodscal) * neq1[k];
           nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
             - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]) +
             (ecprod[2] - prodscal) * neq2[k];
           nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
             - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]) +
             (ecprod[3] - prodscal) * neq3[k];
           nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
             - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]) +
             (ecprod[4] - prodscal) * neq4[k];
           nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
             - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
             - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]) +
             (ecprod[5] - prodscal) * neq5[k];
           nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
             - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
             - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]) +
             (ecprod[6] - prodscal) * neq6[k];
           nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
             - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
             - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]) +
             (ecprod[7] - prodscal) * neq7[k];
           nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
             - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
             - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) +
             (ecprod[8] - prodscal) * neq8[k];
         switch(key_interface)
           {
           case 0: /* nothing */
             break;
           case 2:  /* color field gradient */
             nn = nf0[k] + nf1[k] + nf2[k] + nf3[k] + nf4[k] + nf5[k] + nf6[k] +
               nf7[k] + nf8[k];
            /*
               color_x = sforce * (colorfield[nv1[k]]-colorfield[nv3[k]]) /
               (2.000 * delta_x);
               color_y = sforce * (colorfield[nv2[k]]-colorfield[nv4[k]]) /
               (2.000 * delta_x);
               printf("k=%d  colorx=%e  colory=%e\n",k,color_x,color_y);
             */
            /*
              color_x = sforce * nn * (colorfield[nv1[k]]-colorfield[nv3[k]]) /
              (2.000 * delta_x * mass);
              color_y = sforce * nn * (colorfield[nv2[k]]-colorfield[nv4[k]]) /
              (2.000 * delta_x * mass);
              color_x =  - sforce * nn * gradx[k] / mass;
              color_y =  - sforce * nn * grady[k] / mass;
            */
             color_x = sforce * nn * (colorfield[nv1[k]]-colorfield[nv3[k]]) /
               (2.000 * delta_x * mass);
             color_y = sforce * nn * (colorfield[nv2[k]]-colorfield[nv4[k]]) /
               (2.000 * delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff0[k] += neq0[k] * (ecx[0]*cspeed*color_x + ecy[0]*cspeed*color_y -
                                  prodscal);
             color_x = sforce * nn * (colorfield[k]-colorfield[nv3[k]]) /
               (delta_x * mass);
             color_y = 0.000;
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff1[k] += neq1[k] * (ecx[1]*cspeed*color_x + ecy[1]*cspeed*color_y -
```



```
                                  prodscal);
             color_x = 0.000;
             color_y = sforce * nn * (colorfield[k]-colorfield[nv4[k]]) /
               (delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff2[k] += neq2[k] * (ecx[2]*cspeed*color_x + ecy[2]*cspeed*color_y -
                                  prodscal);
             color_x = sforce * nn * (colorfield[nv1[k]]-colorfield[k]) /
               (delta_x * mass);
             color_y = 0.000;
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff3[k] += neq3[k] * (ecx[3]*cspeed*color_x + ecy[3]*cspeed*color_y -
                                  prodscal);
             color_x = 0.000;
             color_y = sforce * nn * (colorfield[k]-colorfield[nv4[k]]) /
               (delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff4[k] += neq4[k] * (ecx[4]*cspeed*color_x + ecy[4]*cspeed*color_y -
                                  prodscal);
             color_x = sforce * nn * (colorfield[k]-colorfield[nv3[k]]) /
               (delta_x * mass);
             color_y = sforce * nn * (colorfield[k]-colorfield[nv4[k]]) /
               (delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff5[k] += neq5[k] * (ecx[5]*cspeed*color_x + ecy[5]*cspeed*color_y -
                                  prodscal);
             color_x = sforce * nn * (colorfield[nv1[k]]-colorfield[k]) /
               (delta_x * mass);
             color_y = sforce * nn * (colorfield[k]-colorfield[nv4[k]]) /
               (delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff6[k] += neq6[k] * (ecx[6]*cspeed*color_x + ecy[6]*cspeed*color_y -
                                  prodscal);
             color_x = sforce * nn * (colorfield[nv1[k]]-colorfield[k]) /
               (delta_x * mass);
             color_y = sforce * nn * (colorfield[nv2[k]]-colorfield[k]) /
               (delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff7[k] += neq7[k] * (ecx[7]*cspeed*color_x + ecy[7]*cspeed*color_y -
                                  prodscal);
             color_x = sforce * nn * (colorfield[k]-colorfield[nv3[k]]) /
               (delta_x * mass);
             color_y = sforce * nn * (colorfield[nv2[k]]-colorfield[k]) /
               (delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff8[k] += neq8[k] * (ecx[8]*cspeed*color_x + ecy[8]*cspeed*color_y -
                                  prodscal);
             break;

           case 4:  /* color field gradient without nn */
             nn = nf0[k] + nf1[k] + nf2[k] + nf3[k] + nf4[k] + nf5[k] + nf6[k] +
               nf7[k] + nf8[k];

             color_x = sforce * (colorfield[nv1[k]]-colorfield[nv3[k]]) /
               (2.000 * delta_x * mass);
             color_y = sforce * (colorfield[nv2[k]]-colorfield[nv4[k]]) /
               (2.000 * delta_x * mass);
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff0[k] += neq0[k] * (ecx[0]*cspeed*color_x + ecy[0]*cspeed*color_y -
                                  prodscal);
             color_x = sforce * (colorfield[k]-colorfield[nv3[k]]) /
               (delta_x * mass);
             color_y = 0.000;
             prodscal = color_x * uxloc[k] + color_y * uyloc[k];
             nff1[k] += neq1[k] * (ecx[1]*cspeed*color_x + ecy[1]*cspeed*color_y -
                                  prodscal);
             color_x = 0.000;
             color_y = sforce * (colorfield[k]-colorfield[nv4[k]]) /
               (delta_x * mass);
```

```
    prodscal = color_x * uxloc[k] + color_y * uyloc[k];
    nff2[k] += neq2[k] * (ecx[2]*cspeed*color_x + ecy[2]*cspeed*color_y -
                          prodscal);
    color_x = sforce * (colorfield[nv1[k]]-colorfield[k]) /
      (delta_x * mass);
    color_y = 0.000;
    prodscal = color_x * uxloc[k] + color_y * uyloc[k];
    nff3[k] += neq3[k] * (ecx[3]*cspeed*color_x + ecy[3]*cspeed*color_y -
                          prodscal);
    color_x = 0.000;
    color_y = sforce * (colorfield[k]-colorfield[nv4[k]]) /
      (delta_x * mass);
    prodscal = color_x * uxloc[k] + color_y * uyloc[k];
    nff4[k] += neq4[k] * (ecx[4]*cspeed*color_x + ecy[4]*cspeed*color_y -
                          prodscal);
    color_x = sforce * (colorfield[k]-colorfield[nv3[k]]) /
      (delta_x * mass);
    color_y = sforce * (colorfield[k]-colorfield[nv4[k]]) /
      (delta_x * mass);
    prodscal = color_x * uxloc[k] + color_y * uyloc[k];
    nff5[k] += neq5[k] * (ecx[5]*cspeed*color_x + ecy[5]*cspeed*color_y -
                          prodscal);
    color_x = sforce * (colorfield[nv1[k]]-colorfield[k]) /
      (delta_x * mass);
    color_y = sforce * (colorfield[k]-colorfield[nv4[k]]) /
      (delta_x * mass);
    prodscal = color_x * uxloc[k] + color_y * uyloc[k];
    nff6[k] += neq6[k] * (ecx[6]*cspeed*color_x + ecy[6]*cspeed*color_y -
                          prodscal);
    color_x = sforce * (colorfield[nv1[k]]-colorfield[k]) /
      (delta_x * mass);
    color_y = sforce * (colorfield[nv2[k]]-colorfield[k]) /
      (delta_x * mass);
    prodscal = color_x * uxloc[k] + color_y * uyloc[k];
    nff7[k] += neq7[k] * (ecx[7]*cspeed*color_x + ecy[7]*cspeed*color_y -
                          prodscal);
    color_x = sforce * (colorfield[k]-colorfield[nv3[k]]) /
      (delta_x * mass);
    color_y = sforce * (colorfield[nv2[k]]-colorfield[k]) /
      (delta_x * mass);
    prodscal = color_x * uxloc[k] + color_y * uyloc[k];
    nff8[k] += neq8[k] * (ecx[8]*cspeed*color_x + ecy[8]*cspeed*color_y -
                          prodscal);
    break;

  case 3:  /*  HE  */

    color_x = (colorfield[k]-colorfield[nv3[k]]) / delta_x;
    color_y = 0.000;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[1] + color_y * ecy[1]) * cspeed;
    if(colormod)
    nff1[k] += neq1[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[1]*cspeed/2.000) *
       (ecx[1]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[1]*cspeed/2.000) *
       (ecy[1]*cspeed - uyloc[k]));

    color_x = 0.000;
    color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[2] + color_y * ecy[2]) * cspeed;
    if(colormod)
    nff2[k] += neq2[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[2]*cspeed/2.000) *
       (ecx[2]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[2]*cspeed/2.000) *
       (ecy[2]*cspeed - uyloc[k]));
```



```
    color_x = (colorfield[nv1[k]]-colorfield[k]) / delta_x;
    color_y = 0.000;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[3] + color_y * ecy[3]) * cspeed;
    if(colormod)
    nff3[k] += neq3[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[3]*cspeed/2.000) *
       (ecx[3]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[3]*cspeed/2.000) *
       (ecy[3]*cspeed - uyloc[k]));

    color_x = 0.000;
    color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[4] + color_y * ecy[4]) * cspeed;
    if(colormod)
    nff4[k] += neq4[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[4]*cspeed/2.000) *
       (ecx[4]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[4]*cspeed/2.000) *
       (ecy[4]*cspeed - uyloc[k]));

    color_x = (colorfield[k]-colorfield[nv3[k]]) / delta_x;
    color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[5] + color_y * ecy[5]) * cspeed;
    if(colormod)
    nff5[k] += neq5[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[5]*cspeed/2.000) *
       (ecx[5]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[5]*cspeed/2.000) *
       (ecy[5]*cspeed - uyloc[k]));

    color_x = (colorfield[nv1[k]]-colorfield[k]) / delta_x;
    color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[6] + color_y * ecy[6]) * cspeed;
    if(colormod)
    nff6[k] += neq6[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[6]*cspeed/2.000) *
       (ecx[6]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[6]*cspeed/2.000) *
       (ecy[6]*cspeed - uyloc[k]));

    color_x = (colorfield[nv1[k]]-colorfield[k]) / delta_x;
    color_y = (colorfield[nv2[k]]-colorfield[k]) / delta_x;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[7] + color_y * ecy[7]) * cspeed;
    if(colormod)
    nff7[k] += neq7[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[7]*cspeed/2.000) *
       (ecx[7]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[7]*cspeed/2.000) *
       (ecy[7]*cspeed - uyloc[k]));

    color_x = (colorfield[k]-colorfield[nv3[k]]) / delta_x;
    color_y = (colorfield[nv2[k]]-colorfield[k]) / delta_x;
    colormod = sqrt(color_x*color_x + color_y*color_y);
    prodscal = (color_x * ecx[8] + color_y * ecy[8]) * cspeed;
    if(colormod)
    nff8[k] += neq8[k] * (kforce / (2.000 * colormod)) *
      ((color_x*prodscal - colormod*ecx[8]*cspeed/2.000) *
       (ecx[8]*cspeed - uxloc[k]) +
       (color_y*prodscal - colormod*ecy[8]*cspeed/2.000) *
       (ecy[8]*cspeed - uyloc[k]));

    break;
  }
/*
```

```
        printf("k==%d      %g  %g  %g  %g  %g  %g  %g  %g  %g  %g  %g\n",k,
        neq0[k],neq1[k],neq2[k],neq3[k],neq4[k],neq5[k],
        neq6[k],neq7[k],neq8[k],uxloc[k],uyloc[k]);
        printf("k==%d   nf %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
        k,nf0[k],nf1[k],nf2[k],nf3[k],nf4[k],nf5[k],
        nf6[k],nf7[k],nf8[k]);
        printf("k==%d  nff %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
        k,nff0[k],nff1[k],nff2[k],nff3[k],nff4[k],nff5[k],
        nff6[k],nff7[k],nff8[k]);
      */
      break;

       case 1:   /*  bottom  */
         nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
           (ecprod[0] - prodscal) * neq0[k];
         nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
           - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]) +
           (ecprod[1] - prodscal) * neq1[k];
         nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
           - cgrady * ecy[2] * (nf2[nv2[k]] - nf2[k]) +
           (ecprod[2] - prodscal) * neq2[k];
         nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
           - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]) +
           (ecprod[3] - prodscal) * neq3[k];
         nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
           - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]) +
           (ecprod[4] - prodscal) * neq4[k];
         nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
           - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
           - cgrady * ecy[5] * (nf5[nv2[k]] - nf5[k]) +
           (ecprod[5] - prodscal) * neq5[k];
         nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
           - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
           - cgrady * ecy[6] * (nf6[nv2[k]] - nf6[k]) +
           (ecprod[6] - prodscal) * neq6[k];
         nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
           - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
           - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]) +
           (ecprod[7] - prodscal) * neq7[k];
         nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
           - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
           - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) +
           (ecprod[8] - prodscal) * neq8[k];

         break;

       case 2:   /*  top  */
         nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
           (ecprod[0] - prodscal) * neq0[k];
         nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
           - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]) +
           (ecprod[1] - prodscal) * neq1[k];
         nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
           - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]) +
           (ecprod[2] - prodscal) * neq2[k];
         nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
           - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]) +
           (ecprod[3] - prodscal) * neq3[k];
         nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
           - cgrady * ecy[4] * (nf4[k] - nf4[nv4[k]]) +
           (ecprod[4] - prodscal) * neq4[k];
         nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
           - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
           - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]) +
           (ecprod[5] - prodscal) * neq5[k];
         nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
           - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
           - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]) +
           (ecprod[6] - prodscal) * neq6[k];
```



```
         nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
           - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
           - cgrady * ecy[7] * (nf7[k] - nf7[nv4[k]]) +
           (ecprod[7] - prodscal) * neq7[k];
         nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
           - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
           - cgrady * ecy[8] * (nf8[k] - nf8[nv4[k]]) +
           (ecprod[8] - prodscal) * neq8[k];

         break;

       case 3:   /*  left  wall  */

         nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
         nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
           - cgradx * ecx[1] * (nf1[k] - nf1[k]);
         nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
           - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
         nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
           - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]);
         nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
           - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
         nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
           - cgradx * ecx[5] * (nf5[k] - nf5[k])
           - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
         nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
           - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
           - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
         nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
           - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
           - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
         nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
           - cgradx * ecx[8] * (nf8[k] - nf8[k])
           - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
         break;

       case 4:  /*  right  wall */

         nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
         nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
           - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]);
         nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
           - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
         nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
           - cgradx * ecx[3] * (nf3[k] - nf3[k]);
         nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
           - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
         nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
           - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
           - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
         nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
           - cgradx * ecx[6] * (nf6[k] - nf6[k])
           - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
         nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
           - cgradx * ecx[7] * (nf7[k] - nf7[k])
           - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
         nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
           - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
           - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
         break;
       }
    }
}

void first_centered(double mass, double tau, double cspeed,
                    double nf0[], double nf1[], double nf2[],
                    double nf3[], double nf4[], double nf5[],
                    double nf6[], double nf7[], double nf8[],
                    double nff0[], double nff1[], double nff2[],
```

```
                  double nff3[], double nff4[], double nff5[],
                  double nff6[], double nff7[], double nff8[],
                  double neq0[], double neq1[], double neq2[],
                  double neq3[], double neq4[], double neq5[],
                  double neq6[], double neq7[], double neq8[],
                  double ecprod[], double sforce)
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal, color_x, color_y, nn;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                   (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  for(k=0; k<nnodes_all; k++)
    {
     prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k];
     switch(boundary_mode[k])
       {
       case 0:   /*  bulk  */
         nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
           (ecprod[0] - prodscal) * neq0[k];
         nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
           - cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[nv3[k]]) +
           (ecprod[1] - prodscal) * neq1[k];
         nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
           - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]) +
           (ecprod[2] - prodscal) * neq2[k];
         nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
           - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[nv3[k]]) +
           (ecprod[3] - prodscal) * neq3[k];
         nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
           - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]) +
           (ecprod[4] - prodscal) * neq4[k];
         nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
           - cgradx2 * ecx[5] * (nf5[nv1[k]] - nf5[nv3[k]])
           - cgrady2 * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]) +
           (ecprod[5] - prodscal) * neq5[k];
         nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
           - cgradx2 * ecx[6] * (nf6[nv1[k]] - nf6[nv3[k]])
           - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]) +
           (ecprod[6] - prodscal) * neq6[k];
         nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
           - cgradx2 * ecx[7] * (nf7[nv1[k]] - nf7[nv3[k]])
           - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]) +
           (ecprod[7] - prodscal) * neq7[k];
         nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
           - cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[nv3[k]])
           - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]) +
           (ecprod[8] - prodscal) * neq8[k];
      if(gforce)
        {
         /*
         color_x = sforce * (colorfield[nv1[k]]-colorfield[nv3[k]]) /
           (2.000 * delta_x);
         color_y = sforce * (colorfield[nv2[k]]-colorfield[nv4[k]]) /
           (2.000 * delta_x);
         printf("k=%d  colorx=%e  colory=%e\n",k,color_x,color_y);
         */
         nn = nf0[k] + nf1[k] + nf2[k] + nf3[k] + nf4[k] + nf5[k] + nf6[k] +
           nf7[k] + nf8[k];
```



```
         color_x = sforce * nn * (colorfield[nv1[k]]-colorfield[nv3[k]]) /
           (2.000 * delta_x * mass);
         color_y = sforce * nn * (colorfield[nv2[k]]-colorfield[nv4[k]]) /
           (2.000 * delta_x * mass);
         /*
         color_x =  - sforce * nn * gradx[k] / mass;
         color_y =  - sforce * nn * grady[k] / mass;
         */
         prodscal = color_x * uxloc[k] + color_y * uyloc[k];
         nff0[k] += neq0[k] * (ecx[0]*cspeed*color_x + ecy[0]*cspeed*color_y -
                              prodscal);
         nff1[k] += neq1[k] * (ecx[1]*cspeed*color_x + ecy[1]*cspeed*color_y -
                              prodscal);
         nff2[k] += neq2[k] * (ecx[2]*cspeed*color_x + ecy[2]*cspeed*color_y -
                              prodscal);
         nff3[k] += neq3[k] * (ecx[3]*cspeed*color_x + ecy[3]*cspeed*color_y -
                              prodscal);
         nff4[k] += neq4[k] * (ecx[4]*cspeed*color_x + ecy[4]*cspeed*color_y -
                              prodscal);
         nff5[k] += neq5[k] * (ecx[5]*cspeed*color_x + ecy[5]*cspeed*color_y -
                              prodscal);
         nff6[k] += neq6[k] * (ecx[6]*cspeed*color_x + ecy[6]*cspeed*color_y -
                              prodscal);
         nff7[k] += neq7[k] * (ecx[7]*cspeed*color_x + ecy[7]*cspeed*color_y -
                              prodscal);
         nff8[k] += neq8[k] * (ecx[8]*cspeed*color_x + ecy[8]*cspeed*color_y -
                              prodscal);
        }
         break;

       case 1:   /*  bottom  */
         nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
           (ecprod[0] - prodscal) * neq0[k];
         nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
           - cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[nv3[k]]) +
           (ecprod[1] - prodscal) * neq1[k];
         nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
           - cgrady * ecy[2] * (nf2[nv2[k]] - nf2[k]) +
           (ecprod[2] - prodscal) * neq2[k];
         nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
           - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[nv3[k]]) +
           (ecprod[3] - prodscal) * neq3[k];
         nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
           - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]) +
           (ecprod[4] - prodscal) * neq4[k];
         nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
           - cgradx2 * ecx[5] * (nf5[nv1[k]] - nf5[nv3[k]])
           - cgrady * ecy[5] * (nf5[nv2[k]] - nf5[k]) +
           (ecprod[5] - prodscal) * neq5[k];
         nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
           - cgradx2 * ecx[6] * (nf6[nv1[k]] - nf6[nv3[k]])
           - cgrady * ecy[6] * (nf6[nv2[k]] - nf6[k]) +
           (ecprod[6] - prodscal) * neq6[k];
         nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
           - cgradx2 * ecx[7] * (nf7[nv1[k]] - nf7[nv3[k]])
           - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]) +
           (ecprod[7] - prodscal) * neq7[k];
         nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
           - cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[nv3[k]])
           - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) +
           (ecprod[8] - prodscal) * neq8[k];
         break;

       case 2:   /*  top  */
         nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
           (ecprod[0] - prodscal) * neq0[k];
         nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
           - cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[nv3[k]]) +
           (ecprod[1] - prodscal) * neq1[k];
```

```
        nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
          - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]) +
          (ecprod[2] - prodscal) * neq2[k];
        nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
          - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[nv3[k]]) +
          (ecprod[3] - prodscal) * neq3[k];
        nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
          - cgrady * ecy[4] * (nf4[k] - nf4[nv4[k]]) +
          (ecprod[4] - prodscal) * neq4[k];
        nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
          - cgradx2 * ecx[5] * (nf5[nv1[k]] - nf5[nv3[k]])
          - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]) +
          (ecprod[5] - prodscal) * neq5[k];
        nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
          - cgradx2 * ecx[6] * (nf6[nv1[k]] - nf6[nv3[k]])
          - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]) +
          (ecprod[6] - prodscal) * neq6[k];
        nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
          - cgradx2 * ecx[7] * (nf7[nv1[k]] - nf7[nv3[k]])
          - cgrady * ecy[7] * (nf7[k] - nf7[nv4[k]]) +
          (ecprod[7] - prodscal) * neq7[k];
        nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
          - cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[nv3[k]])
          - cgrady * ecy[8] * (nf8[k] - nf8[nv4[k]]) +
          (ecprod[8] - prodscal) * neq8[k];
        break;

      case 3:   /*  left  wall  */

        nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
        nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
          - cgradx * ecx[1] * (nf1[nv1[k]] - nf1[k]);
        nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
          - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
        nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
          - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]);
        nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
          - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
        nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
          - cgradx * ecx[5] * (nf5[nv1[k]] - nf5[k])
          - cgrady2 * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
        nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
          - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
          - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
        nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
          - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
          - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
        nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
          - cgradx * ecx[8] * (nf8[nv1[k]] - nf8[k])
          - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
        break;

      case 4:  /*  right  wall */

        nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
        nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
          - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]);
        nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
          - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
        nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
          - cgradx * ecx[3] * (nf3[k] - nf3[nv3[k]]);
        nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
          - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
        nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
          - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
          - cgrady2 * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
        nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
          - cgradx * ecx[6] * (nf6[k] - nf6[nv3[k]])
          - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
```



```
        nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
          - cgradx * ecx[7] * (nf7[k] - nf7[nv3[k]])
          - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
        nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
          - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
          - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
        break;
      }
    }
}

void second_upwind(double mass, double tau, double cspeed,
                   double nf0[], double nf1[], double nf2[],
                   double nf3[], double nf4[], double nf5[],
                   double nf6[], double nf7[], double nf8[],
                   double nff0[], double nff1[], double nff2[],
                   double nff3[], double nff4[], double nff5[],
                   double nff6[], double nff7[], double nff8[],
                   double neq0[], double neq1[], double neq2[],
                   double neq3[], double neq4[], double neq5[],
                   double neq6[], double neq7[], double neq8[],
                   double ecprod[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                       (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  for(k=0; k<nnodes_all; k++)
    {
      prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k];
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */
          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
            (ecprod[0] - prodscal) * neq0[k];
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau - cgradx2 * ecx[1] *
            (3.000*nf1[k] - 4.000*nf1[nv3[k]] + nf1[nv3[nv3[k]]]) +
            (ecprod[1] - prodscal) * neq1[k];
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady2 * ecy[2] *
            (3.000*nf2[k] - 4.000*nf2[nv4[k]] + nf2[nv4[nv4[k]]]) +
            (ecprod[2] - prodscal) * neq2[k];
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau - cgradx2 * ecx[3] *
            (-3.000*nf3[k] + 4.000*nf3[nv1[k]] - nf3[nv1[nv1[k]]]) +
            (ecprod[3] - prodscal) * neq3[k];
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady2 * ecy[4] *
            (-3.000*nf4[k] + 4.000*nf4[nv2[k]] - nf4[nv2[nv2[k]]]) +
            (ecprod[4] - prodscal) * neq4[k];
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau - cgradx2 * ecx[5] *
            (3.000*nf5[k] - 4.000*nf5[nv3[k]] + nf5[nv3[nv3[k]]]) +
            - cgrady2 * ecy[5] *
            (3.000*nf5[k] - 4.000*nf5[nv4[k]] + nf5[nv4[nv4[k]]]) +
            (ecprod[5] - prodscal) * neq5[k];
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau - cgradx2 * ecx[6] *
            (-3.000*nf6[k] + 4.000*nf6[nv1[k]] - nf6[nv1[nv1[k]]]) +
            - cgrady2 * ecy[6] *
            (3.000*nf6[k] - 4.000*nf6[nv4[k]] + nf6[nv4[nv4[k]]]) +
```

```
          (ecprod[6] - prodscal) * neq6[k];
        nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau - cgradx2 * ecx[7] *
          (-3.000*nf6[k] + 4.000*nf6[nv1[k]] - nf6[nv1[nv1[k]]]) +
          - cgrady2 * ecy[7] *
          (-3.000*nf7[k] + 4.000*nf7[nv2[k]] - nf7[nv2[nv2[k]]]) +
          (ecprod[7] - prodscal) * neq7[k];
        nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau - cgradx2 * ecx[8] *
          (3.000*nf8[k] - 4.000*nf8[nv3[k]] + nf8[nv3[nv3[k]]]) +
          - cgrady2 * ecy[8] *
          (-3.000*nf8[k] + 4.000*nf8[nv2[k]] - nf8[nv2[nv2[k]]]) +
          (ecprod[8] - prodscal) * neq8[k];
        break;
      }
    }
}

void wet9_automaton(void)
{
  int autiter, key_point;
  autiter = 0;

  while( autiter < niter_cycle )
    {
      test_distribution_functions(f10,f11,f12,f13,f14,f15,f16,f17,f18);
      test_distribution_functions(f20,f21,f22,f23,f24,f25,f26,f27,f28);
      compute_local_speeds(f10,f11,f12,f13,f14,f15,f16,f17,f18,
                           f20,f21,f22,f23,f24,f25,f26,f27,f28);
      compute_equilibrium_distributions();
      switch(key_scheme)
        {
        case 0:
          if(key_init > 2)
            compute_local_gradients();
          compute_upwind_sources(f10, f11, f12, f13, f14, f15, f16, f17, f18,
                                 f20, f21, f22, f23, f24, f25, f26, f27, f28);
          upwind_cfl(mass1, tau1, cspeed1, cfl1,
                     f10, f11, f12, f13, f14, f15, f16, f17, f18,
                     ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                     sf10, sf11, sf12, sf13, sf14, sf15, sf16,
                     sf17, sf18,
                     neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                     neq17, neq18,
                     source10, source11, source12, source13, source14,
                     source15, source16, source17, source18);
          upwind_cfl(mass2, tau2, cspeed2, cfl2,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
                     neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                     neq27, neq28,
                     source20, source21, source22, source23, source24,
                     source25, source26, source27, source28);
          break;
        case 1:
          lf_sources(f10, f11, f12, f13, f14, f15, f16, f17, f18,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28);
          lf(sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
             source10, source11, source12, source13, source14,
             source15, source16, source17, source18,
             ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18);
          lf(sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
             source20, source21, source22, source23, source24,
             source25, source26, source27, source28,
             ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28);
          break;
        case 2:
          compute_centered_sources(f10, f11, f12, f13, f14, f15, f16, f17, f18,
                                   f20, f21, f22, f23, f24, f25, f26, f27, f28);
          centered_cfl(cfl1, f10, f11, f12, f13, f14, f15, f16, f17, f18,
```



```
                     ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                     source10, source11, source12, source13, source14,
                     source15, source16, source17, source18);
          centered_cfl(cfl2,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     source20, source21, source22, source23, source24,
                     source25, source26, source27, source28);
          break;
        case 3:
          lw_sources(f10, f11, f12, f13, f14, f15, f16, f17, f18,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28);
          lw(sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
             source10, source11, source12, source13, source14,
             source15, source16, source17, source18,
             ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18);
          lw(sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
             source20, source21, source22, source23, source24,
             source25, source26, source27, source28,
             ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28);
          break;
        case 4:
          islb();
          break;
        case 5:
          islb_linear();
          break;
        case 6:
          islb_upwind();
          break;
        case 7:
        case 8:
          if(key_init > 2)
            compute_local_gradients();
          iprop();
          break;
        case 9:
          ifd();
          break;
        case 10:
          iupfd();
          break;
        case 11:
          iser();
        case 12:
          ilin();
          break;
        case 13:
          if(key_init > 2)
            compute_local_gradients();
          first_upwind(mass1, tau1, cspeed1,
                       f10, f11, f12, f13, f14, f15, f16, f17, f18,
                       ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                       neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                       neq17, neq18, ecprod1, kforce, gradn2x, gradn2y);
          first_upwind(mass2, tau2, cspeed2,
                       f20, f21, f22, f23, f24, f25, f26, f27, f28,
                       ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                       neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                       neq27, neq28, ecprod2, -kforce, gradn1x, gradn1y);
          break;
        case 14:
          if(key_init > 2)
            compute_local_gradients();
          first_centered(mass1, tau1, cspeed1,
                       f10, f11, f12, f13, f14, f15, f16, f17, f18,
                       ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                       neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                       neq17, neq18, ecprod1, kforce);
```

```
      first_centered(mass2, tau2, cspeed2,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                     neq27, neq28, ecprod2, -kforce);
      break;
    case 15:
      second_upwind(mass1, tau1, cspeed1,
                     f10, f11, f12, f13, f14, f15, f16, f17, f18,
                     ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                     neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                     neq17, neq18, ecprod1);
      second_upwind(mass2, tau2, cspeed2,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                     neq27, neq28, ecprod2);
      break;
    }
  autiter++;
  iter++;
  test_distribution_functions(ff10,ff11,ff12,ff13,ff14,ff15,ff16,
                              ff17,ff18);
  test_distribution_functions(ff20,ff21,ff22,ff23,ff24,ff25,ff26,
                              ff27,ff28);
  if(key_scheme < 7)
    {
      compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,ff16,ff17,ff18,
                           ff20,ff21,ff22,ff23,ff24,ff25,ff26,ff27,ff28);
      compute_equilibrium_distributions();
    }
  switch(key_scheme)
    {
    case 0:
      if(key_init > 2)
        compute_local_gradients();
      compute_upwind_sources
        (ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
         ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28);
      upwind_cfl(mass1, tau1, cspeed1, cfl1,
                 ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                 f10, f11, f12, f13, f14, f15, f16, f17, f18,
                 sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
                 neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                 neq17, neq18,
                 source10, source11, source12, source13, source14,
                 source15, source16, source17, source18);
      upwind_cfl(mass2, tau2, cspeed2, cfl2,
                 ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                 f20, f21, f22, f23, f24, f25, f26, f27, f28,
                 sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
                 neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                 neq27, neq28,
                 source20, source21, source22, source23, source24,
                 source25, source26, source27, source28);
      break;
    case 1:
      lf_sources(ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                 ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28);
      lf(sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
         source10, source11, source12, source13, source14,
         source15, source16, source17, source18,
         f10, f11, f12, f13, f14, f15, f16, f17, f18);
      lf(sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
         source20, source21, source22, source23, source24,
         source25, source26, source27, source28,
         f20, f21, f22, f23, f24, f25, f26, f27, f28);
      break;         break;
    case 2:
```



```
      compute_centered_sources
        (ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
         ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28);
      centered_cfl(cfl1,
                   ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                   f10, f11, f12, f13, f14, f15, f16, f17, f18,
                   source10, source11, source12, source13, source14,
                   source15, source16, source17, source18);
      centered_cfl(cfl2,
                   ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                   f20, f21, f22, f23, f24, f25, f26, f27, f28,
                   source20, source21, source22, source23, source24,
                   source25, source26, source27, source28);
      break;
    case 3:
      lw_sources(ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                 ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28);
      lw(sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
         source10, source11, source12, source13, source14,
         source15, source16, source17, source18,
         f10, f11, f12, f13, f14, f15, f16, f17, f18);
      lw(sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
         source20, source21, source22, source23, source24,
         source25, source26, source27, source28,
         f20, f21, f22, f23, f24, f25, f26, f27, f28);
      break;
    case 4:
      islb();
      break;
    case 5:
      islb_linear();
    case 6:
      islb_upwind();
      break;
    case 7:
      ic();
      autiter--;
      iter--;
      break;
    case 8:
      iup();
      autiter--;
      iter--;
      break;
    case 9:
      compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,ff16,ff17,ff18,
                           ff20,ff21,ff22,ff23,ff24,ff25,ff26,ff27,ff28);
      compute_equilibrium_distributions();
      ifdprop();
      break;
    case 10:
      compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,ff16,ff17,ff18,
                           ff20,ff21,ff22,ff23,ff24,ff25,ff26,ff27,ff28);
      compute_equilibrium_distributions();
      ifdprop();
      break;
    case 11:
      ifdprop();
      break;
    case 12:
      ifdprop();
      break;
    case 13:
      if(key_init > 2)
        compute_local_gradients();
      compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,ff16,ff17,ff18,
                           ff20,ff21,ff22,ff23,ff24,ff25,ff26,ff27,ff28);
      compute_equilibrium_distributions();
      first_upwind(mass1, tau1, cspeed1,
```

```
                    ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                    f10, f11, f12, f13, f14, f15, f16, f17, f18,
                    neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                    neq17, neq18, ecprod1, kforce, gradn2x, gradn2y);
        first_upwind(mass2, tau2, cspeed2,
                    ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                    f20, f21, f22, f23, f24, f25, f26, f27, f28,
                    neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                    neq27, neq28, ecprod2, -kforce, gradn1x, gradn1y);
        break;
      case 14:
        if(key_init > 2)
          compute_local_gradients();
        compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,ff16,ff17,ff18,
                             ff20,ff21,ff22,ff23,ff24,ff25,ff26,ff27,ff28);
        compute_equilibrium_distributions();
        first_centered(mass1, tau1, cspeed1,
                    ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                    f10, f11, f12, f13, f14, f15, f16, f17, f18,
                    neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                    neq17, neq18, ecprod1, kforce);
        first_centered(mass2, tau2, cspeed2,
                    ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                    f20, f21, f22, f23, f24, f25, f26, f27, f28,
                    neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                    neq27, neq28, ecprod2, -kforce);
        break;
      case 15:
        second_upwind(mass1, tau1, cspeed1,
                    ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                    f10, f11, f12, f13, f14, f15, f16, f17, f18,
                    neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                    neq17, neq18, ecprod1);
        second_upwind(mass2, tau2, cspeed2,
                    ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                    f20, f21, f22, f23, f24, f25, f26, f27, f28,
                    neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                    neq27, neq28, ecprod2);
        break;
      }
    autiter++;
    iter++;
  }
}


void main(void)
{
  int isim, icycle;
  FILE *frez;

  wet9_input();
  for(isim=0; isim<nsim; isim++)
    {
      nnodes_x = a_nnodes_x[isim];
      nnodes_y = a_nnodes_y[isim];
      nnodes_all = a_nnodes_all[isim];
      lambda = a_lambda[isim];
      key_init = a_key_init[isim];
      key_boundary = a_key_boundary[isim];
      key_scheme = a_key_scheme[isim];
      key_interface = a_key_interface[isim];
      ncycles = a_ncycles[isim];
      niter_cycle = a_niter_cycle[isim];
      niter_init = a_niter_init[isim];
      length_x = a_length_x[isim];
      length_y = a_length_y[isim];
      delta_x = a_delta_x[isim];
      delta_y = a_delta_y[isim];
```



```
      delta_t = a_delta_t[isim];
      force_x = a_force_x[isim];
      force_y = a_force_y[isim];
      gforce = a_gforce[isim];
      mass1 = a_mass1[isim];
      mass2 = a_mass2[isim];
      cspeed1 = a_cspeed1[isim];
      cspeed2 = a_cspeed2[isim];
      tau1 = a_tau1[isim];
      tau2 = a_tau2[isim];
      nzero1 = a_nzero1[isim];
      nzero2 = a_nzero2[isim];
      nzero1left = a_nzero1left[isim];
      nzero2left = a_nzero2left[isim];
      nzero1right = a_nzero1right[isim];
      nzero2right = a_nzero2right[isim];
      uxwall_bot = a_uxwall_bot[isim];
      uywall_bot = a_uywall_bot[isim];
      uxwall_top = a_uxwall_top[isim];
      uywall_top = a_uywall_top[isim];
      uxwall_left = a_uxwall_left[isim];
      uywall_left = a_uywall_left[isim];
      uxwall_right = a_uxwall_right[isim];
      uywall_right = a_uywall_right[isim];

      cspeed12 = cspeed1 * cspeed1;
      cspeed22 = cspeed2 * cspeed2;

      ctau1 = delta_t / tau1;
      ctau2 = delta_t / tau2;
      csforce_x = delta_t * force_x / (kboltz * temp);
      csforce_y = delta_t * force_y / (kboltz * temp);

      kforce = delta_t * gforce / (kboltz * temp);
      kbound1 = delta_t ;
      kbound2 = delta_t ;

      cfl1 = cspeed1 * delta_t / delta_x;
      cfl2 = cspeed2 * delta_t / delta_x;

      oneminus1 = 1.000 - cfl1;
      oneplus1 = 1.000 + cfl1;

      oneminus2 = 1.000 - cfl2;
      oneplus2 = 1.000 + cfl2;

      up1minus2 = cfl1 * (cfl1 - 1.000) / 2.000;
      up1minus1 = cfl1 * (2.000 - cfl1);
      up1 = (2.000 - cfl1) * (1.000 - cfl1) / 2.000;

      up2minus2 = cfl2 * (cfl2 - 1.000) / 2.000;
      up2minus1 = cfl2 * (2.000 - cfl2);
      up2 = (2.000 - cfl2) * (1.000 - cfl2) / 2.000;

      in1 = (1.000 + cfl1) * (2.000 + cfl1) / 2.000;
      in1plus1 = - cfl1 * (2.000 + cfl1);
      in1plus2 = cfl1 * (2.000 + cfl1) / 2.000;

      in2 = (1.000 + cfl2) * (2.000 + cfl2) / 2.000;
      in2plus1 = - cfl2 * (2.000 + cfl2);
      in2plus2 = cfl2 * (2.000 + cfl2) / 2.000;

      t1minus = cfl1 * (1.000 + cfl1) / 2.000;
      t1center = (1.000 - cfl1) * (1.000 + cfl1);
      t1plus = (cfl1 - 1.000) * cfl1 / 2.000;

      t2minus = cfl2 * (1.000 + cfl2) / 2.000;
      t2center = (1.000 - cfl2) * (1.000 + cfl2);
      t2plus = (cfl2 - 1.000) * cfl2 / 2.000;
```

```
        tin1 = (1.000 + cfl1) * (2.000 + cfl1) / 2.000;
        tin1plus1 = - cfl1 * (2.000 + cfl1);
        tin1plus2 = cfl1 * (1.000 + cfl1) / 2.000;

        tin2 = (1.000 + cfl2) * (2.000 + cfl2) / 2.000;
        tin2plus1 = - cfl2 * (2.000 + cfl2);
        tin2plus2 = cfl2 * (1.000 + cfl2) / 2.000;

        tout1 = (1.000 - cfl1 ) * (2.000 - cfl1) / 2.000;
        tout1minus1 = cfl1 * (2.000 - cfl1);
        tout1minus2 = cfl1 * (cfl1 - 1.000) / 2.000;

        tout2 = (1.000 - cfl2 ) * (2.000 - cfl2) / 2.000;
        tout2minus1 = cfl2 * (2.000 - cfl2);
        tout2minus2 = cfl2 * (cfl2 - 1.000) / 2.000;

        tlfminus1 = (1.000 + cfl1) / 2.000;
        tlfplus1 = (1.000 - cfl1) / 2.000;

        tlfminus2 = (1.000 + cfl2) / 2.000;
        tlfplus2 = (1.000 - cfl2) / 2.000;

        isminus1 = cfl1 * (1.000 + cfl1) / 2.000;
        iscenter1 = (1.000 - cfl1) * (1.000 + cfl1);
        isplus1 = cfl1 * (cfl1 - 1.000) / 2.000;

        isminus2 = cfl2 * (1.000 + cfl2) / 2.000;
        iscenter2 = (1.000 - cfl2) * (1.000 + cfl2);
        isplus2 = cfl2 * (cfl2 - 1.000) / 2.000;

        build_names(isim);
        init_lattice_functions();
        init_ecx_nine();
        init_arrays_nine_square();
        init_arrays_nine_square_aux();
        getavec_square();

        iter = 0;
/*
        xv_new(f10,f11,f12,f13,f14,f15,f16,f17,f18,xv_name,1.0001);
        xv(f10,f11,f12,f13,f14,f15,f16,f17,f18,"XV");
*/
        if(key_init > 2)
          {
            xv(f10,f11,f12,f13,f14,f15,f16,f17,f18,"XV1");
            xv(f20,f21,f22,f23,f24,f25,f26,f27,f28,"XV2");
          }
        if(key_init == 3)
          wet9_drop_profile();
        if(key_init == 9)
          wet9_drop_profile();
        test_ntot();

        if(key_init<2)
          wet9_channel_profile();
        if(key_init == 2)
          wet9_couple_profile();

        for(icycle=0; icycle<ncycles; icycle++)
          {
            wet9_automaton();
        if(key_init > 2)
          {
            xv(f10,f11,f12,f13,f14,f15,f16,f17,f18,"XV1");
            xv(f20,f21,f22,f23,f24,f25,f26,f27,f28,"XV2");
          }
        if((key_init == 3) || (key_init == 9))
          wet9_drop_profile();
```



```
            /*
            xv_new(f10,f11,f12,f13,f14,f15,f16,f17,f18,xv_name,1.0001);
            */
            test_ntot();
            if(key_init<2)
              wet9_channel_profile();
            if(key_init == 2)
              wet9_couple_profile();
          }
        /*
        quiver();
        */
        free_lattice_functions();
      }
    free_nsim();
  }
```

```
          nsim= 3
     nnodes_x= 5                    nnodes_y= 25
        lambda= 1
       delta_x= 1.0e-5               delta_y= 0.0001
       delta_t= 1.0e-9
      key_init= 0                key_boundary=  1
    key_scheme= 13
       ncycles= 30                 niter_cycle= 10000          nter_init= 0
         mass1= 1.00                       tau1=  0.00000001
         mass2= 0.90                       tau2=  0.00000001
    nzero1left= 1.00                 nzero2left=  0.00
   nzero1right= 0.00               nzero2right=  0.00
    uxwall_bot= 0.0000000            uywall_bot=  0.000000
    uxwall_top= 0.0000000            uywall_top=  0.000000
   uxwall_left= 0.0000000           uywall_left=  0.000000
  uxwall_right= 0.0000000          uywall_right=  0.000000
       force_x= 0.1000                  force_y=  0.0000    gforce= 00000000.000
      nnodes_x= 5                    nnodes_y= 25
        lambda= 1
       delta_x= 5.0e-5               delta_y= 0.0001
       delta_t= 1.0e-9
      key_init= 0                key_boundary=  1
    key_scheme= 13
       ncycles= 50                 niter_cycle= 10000          nter_init= 0
         mass1= 1.00                       tau1=  0.00000001
         mass2= 0.90                       tau2=  0.00000001
    nzero1left= 1.00                 nzero2left=  0.00
   nzero1right= 0.00               nzero2right=  0.00
    uxwall_bot= 0.0000000            uywall_bot=  0.000000
    uxwall_top= 0.0000000            uywall_top=  0.000000
   uxwall_left= 0.0000000           uywall_left=  0.000000
  uxwall_right= 0.0000000          uywall_right=  0.000000
       force_x= 0.1000                  force_y=  0.0000    gforce= 00000000.000
      nnodes_x= 5                    nnodes_y= 25
        lambda= 1
       delta_x= 10.0e-5               delta_y= 0.0001
       delta_t= 1.0e-9
      key_init= 0                key_boundary=  1
    key_scheme= 13
       ncycles= 60                 niter_cycle= 10000          nter_init= 0
         mass1= 1.00                       tau1=  0.00000001
         mass2= 0.90                       tau2=  0.00000001
    nzero1left= 1.00                 nzero2left=  0.00
   nzero1right= 0.00               nzero2right=  0.00
    uxwall_bot= 0.0000000            uywall_bot=  0.000000
    uxwall_top= 0.0000000            uywall_top=  0.000000
   uxwall_left= 0.0000000           uywall_left=  0.000000
  uxwall_right= 0.0000000          uywall_right=  0.000000
       force_x= 0.1000                  force_y=  0.0000    gforce= 00000000.000
```

Appendix B

dif9 code

```c
/*****************************************************************\
*                                                                 *
*  dif9head.h  -- definition of global variables                  *
*                                                                 *
\*****************************************************************/

#ifdef MAIN_HEADER

  const double kboltz=1.381e-23, amu=1.661e-27, temp=300.00;

/*  kboltz = J/(molecules*K), amu = (kg), temp = (K);  */

  const double w0 = ((double) 4) / ((double) 9),
               w1 = ((double) 1) / ((double) 9),
               w2 = ((double) 1) / ((double) 36);

  const double three = ((double) 3),
               three_over_two = ((double) 3) / ((double) 2),
               nine_over_two  = ((double) 9) / ((double) 2);

  const double uxwall = 0.00000, uywall = 0.00000;

  int nnodes_x, nnodes_y, nnodes_all, lambda;

  int *a_nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda;

  int key_init, key_boundary, key_g, key_force, key_point, key_scheme;

  int *a_key_init, *a_key_boundary, *a_key_g, *a_key_force, *a_key_scheme;

  int nsim, ncycles, niter_cycle, niter_init, iter, niter;

  int *a_ncycles, *a_niter_cycle, *a_niter_init;

  double length_x, length_y, delta_x, delta_y, delta_t,
         cspeed1, cspeed2, cspeed12, cspeed22,
         cspeed1s, cspeed2s, cspeed1s2, cspeed2s2,
         force_x, force_y;

  double *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_t,
         *a_force_x, *a_force_y;

  double mass1, mass2, tau1, tau2, dcoef, alpha, alpha1;

  double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2,
         *a_dcoef;

  double nzero1left, nzero2left, nzero1right, nzero2right;

  double *a_nzero1left, *a_nzero2left, *a_nzero1right, *a_nzero2right;

  char input_name[] = "dif9.input", output_name[] = "dif9.output";

  char id_name[128], rez_name[128], xv_name[128];

  double ecx[9], ecy[9], ecxx[9], ecxy[9], ecyx[9], ecyy[9], edxy;

  double ecx1[9], ecy1[9], ecxx1[9], ecxy1[9], ecyx1[9], ecyy1[9];

  double ecx2[9], ecy2[9], ecxx2[9], ecxy2[9], ecyx2[9], ecyy2[9];

  double ec1[9], ec2[9];

  int *boundary_mode;

  double *f10, *f11, *f12, *f13, *f14, *f15, *f16, *f17, *f18;

  double *f20, *f21, *f22, *f23, *f24, *f25, *f26, *f27, *f28;
```



```c
  double *ff10, *ff11, *ff12, *ff13, *ff14, *ff15, *ff16, *ff17, *ff18;

  double *ff20, *ff21, *ff22, *ff23, *ff24, *ff25, *ff26, *ff27, *ff28;

  double *sf10, *sf11, *sf12, *sf13, *sf14, *sf15, *sf16, *sf17, *sf18;

  double *sf20, *sf21, *sf22, *sf23, *sf24, *sf25, *sf26, *sf27, *sf28;

  double *neq10,*neq11,*neq12,*neq13,*neq14,*neq15,*neq16,*neq17,*neq18;

  double *neq20,*neq21,*neq22,*neq23,*neq24,*neq25,*neq26,*neq27,*neq28;

  double *uxloc, *uyloc, *uxloc1, *uyloc1, *uxloc2, *uyloc2;

  int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;

  double *nloc1, *nloc2, *rholoc1, *rholoc2, *rholoc;

#else

  extern const double kboltz, amu, temp;

  extern const double w0, w1, w2;

  extern const double three, three_over_two, nine_over_two;

  extern const double uxwall, uywall;

  extern int nnodes_x, nnodes_y, nnodes_all, lambda;

  extern int *a_nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda;

  extern int key_init, key_boundary, key_g, key_force, key_point, key_scheme;

  extern int *a_key_init, *a_key_boundary, *a_key_g, *a_key_force,
         *a_key_scheme;

  extern int nsim, ncycles, niter_cycle, niter_init, iter, niter;

  extern int *a_ncycles, *a_niter_cycle, *a_niter_init;

  extern double length_x, length_y, delta_x, delta_y, delta_t,
                cspeed1, cspeed2, cspeed12, cspeed22,
                cspeed1s, cspeed2s, cspeed1s2, cspeed2s2,
                force_x, force_y;

  extern double *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_t,
                *a_force_x, *a_force_y;

  extern double mass1, mass2, tau1, tau2, dcoef, alpha, alpha1;

  extern double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2,
                *a_dcoef;

  extern double nzero1left, nzero2left, nzero1right, nzero2right;

  extern double *a_nzero1left, *a_nzero2left, *a_nzero1right, *a_nzero2right;

  extern char input_name[], output_name[];

  extern char id_name[], rez_name[], xv_name[];

  extern double ecx[], ecy[], ecxx[], ecxy[], ecyx[], ecyy[], edxy;

  extern double ecx1[], ecy1[], ecxx1[], ecxy1[], ecyx1[], ecyy1[];

  extern double ecx2[], ecy2[], ecxx2[], ecxy2[], ecyx2[], ecyy2[];

  extern double ec1[], ec2[];
```

```
  extern int *boundary_mode;
  extern double *f10, *f11, *f12, *f13, *f14, *f15, *f16, *f17, *f18;
  extern double *f20, *f21, *f22, *f23, *f24, *f25, *f26, *f27, *f28;
  extern double *ff10, *ff11, *ff12, *ff13, *ff14, *ff15, *ff16, *ff17, *ff18;
  extern double *ff20, *ff21, *ff22, *ff23, *ff24, *ff25, *ff26, *ff27, *ff28;
  extern double *sf10, *sf11, *sf12, *sf13, *sf14, *sf15, *sf16, *sf17, *sf18;
  extern double *sf20, *sf21, *sf22, *sf23, *sf24, *sf25, *sf26, *sf27, *sf28;
  extern double *neq10,*neq11,*neq12,*neq13,*neq14,*neq15,*neq16,*neq17,*neq18;
  extern double *neq20,*neq21,*neq22,*neq23,*neq24,*neq25,*neq26,*neq27,*neq28;
  extern double *uxloc, *uyloc, *uxloc1, *uyloc1, *uxloc2, *uyloc2; ;
  extern int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;
  extern double *nloc1, *nloc2, *rholoc1, *rholoc2, *rholoc;
#endif
```

```c
/**********************************************************************\
*                                                                      *
*  dif9inout.c                                                         *
*                                                                      *
\**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "dif9head.h"

void allocate_nsim(void)
{
  a_nnodes_x = (int*) malloc(nsim*sizeof(int));
  a_nnodes_y = (int*) malloc(nsim*sizeof(int));
  a_nnodes_all = (int*) malloc(nsim*sizeof(int));
  a_lambda = (int*) malloc(nsim*sizeof(int));
  a_key_init = (int*) malloc(nsim*sizeof(int));
  a_key_boundary = (int*) malloc(nsim*sizeof(int));
  a_key_g = (int*) malloc(nsim*sizeof(int));
  a_key_force = (int*) malloc(nsim*sizeof(int));
  a_key_scheme = (int*) malloc(nsim*sizeof(int));
  a_ncycles = (int*) malloc(nsim*sizeof(int));
  a_niter_cycle = (int*) malloc(nsim*sizeof(int));
  a_niter_init = (int*) malloc(nsim*sizeof(int));
  a_length_x = (double*) malloc(nsim*sizeof(double));
  a_length_y = (double*) malloc(nsim*sizeof(double));
  a_delta_x = (double*) malloc(nsim*sizeof(double));
  a_delta_y = (double*) malloc(nsim*sizeof(double));
  a_delta_t = (double*) malloc(nsim*sizeof(double));
  a_force_x = (double*) malloc(nsim*sizeof(double));
  a_force_y = (double*) malloc(nsim*sizeof(double));
  a_mass1 = (double*) malloc(nsim*sizeof(double));
  a_mass2 = (double*) malloc(nsim*sizeof(double));
  a_cspeed1 = (double*) malloc(nsim*sizeof(double));
  a_cspeed2 = (double*) malloc(nsim*sizeof(double));
  a_tau1 =  (double*) malloc(nsim*sizeof(double));
  a_tau2 =  (double*) malloc(nsim*sizeof(double));
  a_dcoef =  (double*) malloc(nsim*sizeof(double));
  a_nzero1left =  (double*) malloc(nsim*sizeof(double));
  a_nzero2left =  (double*) malloc(nsim*sizeof(double));
  a_nzero1right =  (double*) malloc(nsim*sizeof(double));
  a_nzero2right =  (double*) malloc(nsim*sizeof(double));
}

void free_nsim(void)
{
  free(a_nnodes_x);
  free(a_nnodes_y);
  free(a_nnodes_all);
  free(a_lambda);
  free(a_key_init);
  free(a_key_boundary);
  free(a_key_g);
  free(a_key_force);
  free(a_key_scheme);
  free(a_ncycles);
  free(a_niter_cycle);
  free(a_niter_init);
  free(a_length_x);
  free(a_length_y);
  free(a_delta_x);
  free(a_delta_y);
  free(a_delta_t);
  free(a_force_x);
  free(a_force_y);
  free(a_mass1);
```



```c
  free(a_mass2);
  free(a_cspeed1);
  free(a_cspeed2);
  free(a_tau1);
  free(a_tau2);
  free(a_dcoef);
  free(a_nzero1left);
  free(a_nzero2left);
  free(a_nzero1right);
  free(a_nzero2right);
}

void dif9_input(void)
{
  int i, j;
  char dummy[128];
  /*
    double kboltz=1.381e-23, amu=1.661e-27, temp=300.00;
    kboltz = J/(molecules*K), amu = (kg), temp - (K);
  */
  FILE *fin, *frez;

  frez=fopen(output_name,"w");
  if((fin = fopen(input_name,"r")) == NULL)
  {
    fprintf(frez,"program <dif9> stopped - input file <%s> does not exist !\n",
            input_name);
    fclose(frez);
    exit(1);
  }
  fscanf(fin,"%s %d\n",dummy,&nsim);
  if(nsim < 1)
    exit(1);

  allocate_nsim();

  for(i=0; i<nsim; i++)
    {
      fscanf(fin,"%s %d %s %d\n",dummy,&nnodes_x,dummy,&nnodes_y);
      fprintf(frez,"nnodes_x= %d  nnodes_y= %d\n",nnodes_x,nnodes_y);
      fscanf(fin,"%s %d\n",dummy,&lambda);
      fprintf(frez,"lambda= %d\n",lambda);
      fscanf(fin,"%s %lf %s %lf %s %lf\n",
             dummy,&delta_x,dummy,&delta_y,dummy,&delta_t);
      fprintf(frez,"delta_x= %lf  delta_y= %lf  delta_t= %20.15lf\n",
             delta_x,delta_y,delta_t);
      fscanf(fin,"%s %d %s %d %s %d %s %d\n",
             dummy,&key_init,dummy,&key_boundary,dummy,&key_force,
             dummy,&key_scheme);
      fprintf(frez,"key_init= %d  key_boundary= %d  key_force= %d  key_scheme=%d\n",
             key_init,key_boundary,key_force,key_scheme);
      fscanf(fin,"%s %d %s %d %s %d\n",
             dummy,&ncycles,dummy,&niter_cycle,dummy,&niter_init);
      fprintf(frez,"ncycles= %d  niter_cycle= %d  niter_init= %d\n",
             ncycles,niter_cycle,niter_init);
      fscanf(fin,"%s %lf %s %lf\n",dummy,&mass1,dummy,&tau1);
      fscanf(fin,"%s %lf %s %lf\n",dummy,&mass2,dummy,&tau2);
      fscanf(fin,"%s %lf %s %lf\n",dummy,&nzero1left,dummy,&nzero1right);
      fscanf(fin,"%s %lf %s %lf\n",dummy,&nzero2left,dummy,&nzero2right);
      fscanf(fin,"%s %lf\n",dummy,&dcoef);
      fprintf(frez,"mass1=%lf  tau1=%lf\n",mass1,tau1);
      fprintf(frez,"mass2=%lf  tau2=%lf\n",mass2,tau2);
      fprintf(frez,"nzero1left=%lf  nzero1right=%lf\n",nzero1left,nzero1right);
      fprintf(frez,"nzero2left=%lf  nzero2right=%lf\n",nzero2left,nzero2right);
      fprintf(frez,"dcoef=%lf\n",dcoef);
      fscanf(fin,"%s %lf %s %lf\n",dummy,&force_x,dummy,&force_y);
      force_x *= 1.0e-20;
      force_y *= 1.0e-20;
      fprintf(frez,"force_x=%lf  force_y=%lf\n",force_x,force_y);
```

```
   if(key_scheme == 7)
     key_boundary += 5;

   cspeed1 = (4.00 * kboltz * temp) / (mass1 * amu);
   cspeed2 = (4.00 * kboltz * temp) / (mass2 * amu);
   cspeed1 = sqrt(cspeed1);
   cspeed2 = sqrt(cspeed2);

   if( (((double) lambda)*delta_t*cspeed1 > delta_x) ||
       (((double) lambda)*delta_t*cspeed1 > delta_y) ||
       (((double) lambda)*delta_t*cspeed2 > delta_x) ||
       (((double) lambda)*delta_t*cspeed2 > delta_y) )
     {
       printf("<dif9> stopped - small delta_x or delta_y !\n");
       fprintf(frez,"<dif9> stopped - small delta_x or delta_y !\n");
       fclose(frez);
       exit(1);
     }

   if( (delta_t > tau1) || (delta_t > tau2) )
     {
       printf("<dif9> stopped - delta_t > tau 1 or tau2 !\n");
       fprintf(frez,"<dif9> stopped - delta_t > tau 1 or tau2 !\n");
       fclose(frez);
       exit(1);
     }

   nnodes_all = nnodes_x * nnodes_y;
   length_x = delta_x * ((double) (nnodes_x-1));
   length_y = delta_y * ((double) (nnodes_y-1));

   fprintf(frez,"%d  %d  %d  %lf  %lf  %lf\n",nnodes_x,nnodes_y,
           nnodes_all,length_x,length_y,delta_t);
   fprintf(frez,"kboltz= %e  %lf %lf %lf %e\n",kboltz,temp,mass1,mass2,amu);

   a_nnodes_x[i] = nnodes_x;
   a_nnodes_y[i] = nnodes_y;
   a_nnodes_all[i] = nnodes_all;
   a_lambda[i] = lambda;
   a_key_init[i] = key_init;
   a_key_boundary[i] = key_boundary;
   a_key_force[i] = key_force;
   a_key_scheme[i] = key_scheme;
   a_ncycles[i] = ncycles;
   a_niter_cycle[i] = niter_cycle;
   a_niter_init[i] = niter_init;
   a_length_x[i] = length_x;
   a_length_y[i] = length_y;
   a_delta_x[i] = delta_x;
   a_delta_y[i] = delta_y;
   a_delta_t[i] = delta_t;
   a_force_x[i] = force_x;
   a_force_y[i] = force_y;
   a_mass1[i] = mass1;
   a_mass2[i] = mass2;
   a_cspeed1[i] = cspeed1;
   a_cspeed2[i] = cspeed2;
   a_tau1[i] = tau1;
   a_tau2[i] = tau2;
   a_dcoef[i] = dcoef;
   a_nzero1left[i] = nzero1left;
   a_nzero2left[i] = nzero2left;
   a_nzero1right[i] = nzero1right;
   a_nzero2right[i] = nzero2right;
  }
 fclose(fin);
 fclose(frez);
}
```



```
void dif9_output(void)
{
  int i;
  FILE *fout;
  fout = fopen(output_name,"w");
  fprintf(fout,"nsim= %d\n",nsim);
  for(i=0; i<nsim; i++)
    {
      fprintf(fout,"nnodes_x= %d  nnodes_y= %d\n",a_nnodes_x[i],a_nnodes_y[i]);
      fprintf(fout,"delta_x= %lf  delta_y= %lf  delta_t= %lf\n",
              a_delta_x[i],a_delta_y[i],a_delta_t[i]);
      fprintf(fout,"key_init= %d  key_boumdary= %d  key_force= %d\n",
              a_key_init[i],a_key_boundary[i],a_key_force[i]);
      fprintf(fout,"ncycles= %d  niter_cycle= %d  niter_init= %d\n",
              a_ncycles[i],a_niter_cycle[i],a_niter_init[i]);
      fprintf(fout,"mass1= %lf tau1= %lf\n",a_mass1[i],a_tau1[i]);
      fprintf(fout,"mass2= %lf tau2= %lf\n",a_mass2[i],a_tau2[i]);
      fprintf(fout,"force_x= %lf  force_y= %lf\n",a_force_x[i],a_force_y[i]);
    }
  fclose(fout);
}

void build_names()
{
  int imass1,imass2,itau1,itau2,icspeed1,icspeed2,
    idelta_x,idelta_y,idelta_t,iforce_x,iforce_y,idcoef,
    inzero1left,inzero2left,inzero1right,inzero2right;
  imass1 = (int) (mass1 * 100);
  imass2 = (int) (mass2 * 100);
  itau1 = (int) (tau1 * 1.00e+12);
  itau2 = (int) (tau2 * 1.00e+12);
  icspeed1 = (int) cspeed1;
  icspeed2 = (int) cspeed2;
  idelta_x = (int) (delta_x * 1.00e+10);
  idelta_y = (int) (delta_y * 1.00e+10);
  idelta_t = (int) (delta_t * 1.00e+12);
  iforce_x = (int) (force_x * 1.00e+25);
  iforce_y = (int) (force_y * 1.00e+25);
  idcoef = (int) (dcoef * 1.00e+10);
  inzero1left = (int) (nzero1left * 100.000);
  inzero2left = (int) (nzero2left * 100.000);
  inzero1right = (int) (nzero1right * 100.000);
  inzero2right = (int) (nzero2right * 100.000);
  printf("%lf\n",dcoef);
  sprintf(id_name,
     "_%1d%1d%1d%1d_%3.3d_%3.3d_%3.3d_%4.4d_%5.5d_%3.3d_%4.4d_%5.5d_%5.5d_%2.2d_%9.9d_%9
.9d_%6.6d_%3.3d_%3.3d_%3.3d_%3.3d",
          key_scheme,key_init,key_boundary,key_force,
          nnodes_x,nnodes_y,imass1,icspeed1,itau1,
          imass2,icspeed2,itau2,idelta_t,lambda,idelta_x,idelta_y,idcoef,
          inzero1left,inzero1right,inzero2left,inzero2right);
          sprintf(rez_name,"R%s",id_name);
          sprintf(xv_name,"X%s",id_name);
}
```

```c
/*********************************************************************\
*                                                                     *
*  dif9init.c  --  arrays initialisation                              *
*                                                                     *
\*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "dif9head.h"

void print_tavec(int kk, int nv[])
{
  int i,j;
  FILE *fout;
  fout = fopen(rez_name,"aw");
  fprintf(fout,"\nnv%1d\n",kk);
  for(j=0;j<=nnodes_y;j++)
    {
      for(i=0;i<=nnodes_x;i++)
        {
          fprintf(fout,"%3d ",nv[(j-1)*nnodes_x+i]);
        }
      fprintf(fout,"\n");
    }
  fclose(fout);
}

void getavec_square(void)
{
  int i, j, k;
  k = 0;
  for(j=0; j<nnodes_y; j++)
    {
      for(i=0; i<nnodes_x; i++)
        {
          nv1[k] = k+1;
          if(i == (nnodes_x-1))
            nv1[k] -= nnodes_x;
          nv2[k] = k+nnodes_x;
          if(nv2[k] >= nnodes_all)
            nv2[k] -= nnodes_all;
          nv3[k] = k-1;
          if(i == 0)
            nv3[k] += nnodes_x;
          nv4[k] = k-nnodes_x;
          if(nv4[k] < 0)
            nv4[k]+= nnodes_all;
          nv5[k] = nv2[k] + 1;
          if(i == (nnodes_x-1))
            nv5[k] -= nnodes_x;
          nv6[k] = nv2[k] - 1;
          if(i == 0)
            nv6[k] += nnodes_x;
          nv7[k] = nv4[k] - 1;
          if(i == 0)
            nv7[k] += nnodes_x;
          nv8[k] = nv4[k] + 1;
          if(i == (nnodes_x-1))
            nv8[k] -= nnodes_x;
          /*
          printf("k=%d  %d  %d  %d  %d  %d  %d  %d  %d\n",
                k,nv1[k],nv2[k],nv3[k],nv4[k],nv5[k],nv6[k],nv7[k],nv8[k]);
          */
          k++;
        }
    }
```



```c
}

void init_ecx_nine(void)
{
  int i;

  ecx[0]=((double) 0);
  ecy[0]=((double) 0);
  ecx[1]=((double) 1);
  ecy[1]=((double) 0);
  ecx[2]=((double) 0);
  ecy[2]=((double) 1);
  ecx[3]=((double) -1);
  ecy[3]=((double) 0);
  ecx[4]=((double) 0);
  ecy[4]=((double) -1);
  ecx[5]=((double) 1);
  ecy[5]=((double) 1);
  ecx[6]=((double) -1);
  ecy[6]=((double) 1);
  ecx[7]=((double) -1);
  ecy[7]=((double) -1);
  ecx[8]=((double) 1);
  ecy[8]=((double) -1);

  for( i=0; i < 9; i++)
    {
      ecxx[i] = ecx[i]*ecx[i];
      ecxy[i] = 2.0e+0*ecx[i]*ecy[i];
      ecyx[i] = 2.0e+0*ecy[i]*ecx[i];
      ecyy[i] = ecy[i]*ecy[i];

      ecx1[i] = ecx[i] * cspeed1;
      ecy1[i] = ecy[i] * cspeed1;
      ecxx1[i] = ecxx[i] * cspeed1 * cspeed1;
      ecxy1[i] = ecxy[i] * cspeed1 * cspeed1;
      ecyx1[i] = ecyx[i] * cspeed1 * cspeed1;
      ecyy1[i] = ecyy[i] * cspeed1 * cspeed1;

      ecx2[i] = ecx[i] * cspeed2;
      ecy2[i] = ecy[i] * cspeed2;
      ecxx2[i] = ecxx[i] * cspeed2 * cspeed2;
      ecxy2[i] = ecxy[i] * cspeed2 * cspeed2;
      ecyx2[i] = ecyx[i] * cspeed2 * cspeed2;
      ecyy2[i] = ecyy[i] * cspeed2 * cspeed2;

      ec1[i] = ecx1[i] * sqrt((double) 2);
      ec2[i] = ecx2[i] * sqrt((double) 2);
    }
  edxy = sqrt((double) 2);
}

void init_arrays_nine_square(void)
{
  FILE *fsav, *frez;
  int i,j,k;
  double rand_coef = 0.00000001, nzero = 1.00, dummy;

  frez=fopen(rez_name,"a");

  for( i=0; i<9; i++)
    {
      fprintf(frez,"i,ecx[i],ecy[i]\n%3d %lf %lf\n",
              i,ecx[i],ecy[i]);
      fprintf(frez,
              "i,ecxx[i],ecxy[i],ecyx[i],ecyy[i]\n%3d %lf %lf %lf %lf\n",
              i,ecxx[i],ecxy[i],ecyx[i],ecyy[i]);
      fprintf(frez,"i,ecx1[i],ecy1[i]\n%3d %lf %lf\n",
              i,ecx1[i],ecy1[i]);
```

```
      fprintf(frez,"i,ecx2[i],ecy2[i]\n%3d %lf %lf\n",
              i,ecx2[i],ecy2[i]);
    }


  switch (key_init)
    {
    case 0: /* horizontal diffusion couple */
      k=-1;
      for ( j=0; j< nnodes_y; j++)
        {
          for( i=0; i < nnodes_x; i++)
            {
              k++;
              if(i<=nnodes_x/2)
                {
                  f10[k] = nzero * (1. +rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = 0.0e+0;
                  /*
                    f20[k] = (nzero*0.9) * (1.0 +rand_coef *
                    ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  */
                }
              else
                {
                  /*
                    f10[k] = (nzero*0.9) * (1. +rand_coef *
                    ( ((double) rand() / (double) RAND_MAX) - 0.5));
                    */
                  f10[k] = 0.0e+0;
                  f20[k] = nzero * (1.0 +rand_coef *
                            ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  /*
                  f20[k] = 0.0e+0;
                  f10[k] = (nzero*0.9) * (1. +rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  */
                }
            }
        }
      break;
    case 1: /* horizontal one component self - diffusion  */
      k=-1;
      for ( j=0; j< nnodes_y; j++)
        {
          for( i=0; i < nnodes_x; i++)
            {
              k++;
              if(i<=nnodes_x/2)
                {
                  f10[k] = nzero * (1. +rand_coef *
                           ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = 0.0e+0;
                  /*
                    f20[k] = (nzero*0.9) * (1.0 +rand_coef *
                    ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  */
                }
              else
                {
                  /*
                    f10[k] = (nzero*0.9) * (1. +rand_coef *
                    ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f20[k] = nzero * (1.0 +rand_coef *
                            ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  f10[k] = 0.0e+0;
                  */
                  f20[k] = 0.0e+0;
```



```
                    f10[k] = (nzero*0.1) * (1. +rand_coef *
                    ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  }
              }
          }
        break;
      case 8: /* horizontal diffusion couple, erf */
        k=-1;
        for ( j=0; j< nnodes_y; j++)
          {
            for( i=0; i < nnodes_x; i++)
              {
                k++;
                f10[k] = nzero * (1. +rand_coef *
                         ( ((double) rand() / (double) RAND_MAX) - 0.5));
                f20[k] = nzero * (1.0 +rand_coef *
                         ( ((double) rand() / (double) RAND_MAX) - 0.5));
                f10[k] *=  (1.000 - 0.5000 * ( 1.000 + erf(0.01000*((double)
                                               (i-nnodes_x/2 -1))) ));
                f20[k] *=  (0.5000 * ( 1.000 + erf(0.01000*((double)
                                       (i-nnodes_x/2-1))) ));
              }
          }
        break;
      case 9: /* horizontal diffusion couple, linear */
        k=-1;
        for ( j=0; j< nnodes_y; j++)
          {
            for( i=0; i < nnodes_x; i++)
              {
                k++;
                if(i<=nnodes_x/4)
                  {
                    f10[k] = nzero * (1. +rand_coef *
                             ( ((double) rand() / (double) RAND_MAX) - 0.5));
                    f20[k] = 0.0e+0;
                    /*
                      f20[k] = (nzero*0.9) * (1.0 +rand_coef *
                      ( ((double) rand() / (double) RAND_MAX) - 0.5));
                    */
                  }
                if((i>nnodes_x/4)&&(i<=3*nnodes_x/4))
                  {
                    f10[k] = nzero * (1. +rand_coef *
                             ( ((double) rand() / (double) RAND_MAX) - 0.5));
                    f20[k] = nzero * (1.0 +rand_coef *
                              ( ((double) rand() / (double) RAND_MAX) - 0.5));
                    f10[k] *= ((double)(3 * nnodes_x / 4 - i)) / (nnodes_x / 2);
                    f20[k] *= ((double)(i - nnodes_x / 4)) / (nnodes_x / 2);
                  }
                if(i>3*nnodes_x/4)
                  {
                    /*
                    f10[k] = (nzero*0.9) * (1. +rand_coef *
                             ( ((double) rand() / (double) RAND_MAX) - 0.5));
                    */
                    f10[k] = 0.0e+0;
                    f20[k] = nzero * (1.0 +rand_coef *
                              ( ((double) rand() / (double) RAND_MAX) - 0.5));
                  }
              }
          }
        break;
      case 2: /* vertical diffusion couple */
        k=-1;
        for ( j=0; j< nnodes_y; j++)
          {
            for( i=0; i < nnodes_x; i++)
              {
```

```
          k++;
          if(j<=nnodes_y/2)
            {
              f10[k] = nzero * (1. +rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
              f20[k] = 0.0e+0;
            }
          else
            {
              f10[k] = 0.0e+0;
              f20[k] = nzero * (1. +rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
            }
        }
      }
    break;
  case 5: /* horizontal diffusion couple with variable nleft, nright*/
    k=-1;
    for ( j=0; j< nnodes_y; j++)
      {
        for( i=0; i < nnodes_x; i++)
          {
          k++;
          if(i<=nnodes_x/2)
            {
              f10[k] = nzero1left * (1. +rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
              f20[k] = nzero2left * (1. +rand_coef *
                       ( ((double) rand() / (double) RAND_MAX) - 0.5));
            }
          else
            {
              f10[k] = nzero1right * (1.0 +rand_coef *
                        ( ((double) rand() / (double) RAND_MAX) - 0.5));
              f20[k] = nzero2right * (1.0 +rand_coef *
                        ( ((double) rand() / (double) RAND_MAX) - 0.5));
            }
          }
      }
    break;
  }

 for( k=0; k < nnodes_all; k++)
   {
     f11[k] = w1 * f10[k];
     f12[k] = w1 * f10[k];
     f13[k] = w1 * f10[k];
     f14[k] = w1 * f10[k];
     f15[k] = w2 * f10[k];
     f16[k] = w2 * f10[k];
     f17[k] = w2 * f10[k];
     f18[k] = w2 * f10[k];
     f10[k] *= w0;
     f21[k] = w1 * f20[k];
     f22[k] = w1 * f20[k];
     f23[k] = w1 * f20[k];
     f24[k] = w1 * f20[k];
     f25[k] = w2 * f20[k];
     f26[k] = w2 * f20[k];
     f27[k] = w2 * f20[k];
     f28[k] = w2 * f20[k];
     f20[k] *= w0;
   }

 fclose(frez);
}

void init_arrays_nine_square_aux(void)
{
```



```
  int i, j, nnl1 = (nnodes_y-1)*nnodes_x;
  switch(key_boundary)
    {
    case 0:  /*  periodic boundaries  -  finite difference model  */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 0;
      break;
    case 1:  /*  horizontal walls  -  finite difference model  */
      for ( i = 0; i< nnodes_x; i++)
        boundary_mode[i] = 1;
      for ( i = nnodes_x; i < nnodes_all-nnodes_x; i++)
        boundary_mode[i] = 0;
      for ( i = nnodes_all-nnodes_x; i < nnodes_all; i++)
        boundary_mode[i] = 2;
      break;
    case 2:  /*  vertical walls  -  finite difference model  */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 0;
      for (j=0; j < nnodes_y; j++)
        {
          boundary_mode[j*nnodes_x] = 3;
          boundary_mode[(j+1)*nnodes_x-1] = 4;
          /*
          boundary_mode[j*nnodes_x+1] = 31;
          boundary_mode[(j+1)*nnodes_x-2] = 41;
          */
        }
      break;
    case 3:  /*  container  -  finite difference model  */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 0;
      for ( i = 0; i< nnodes_x; i++)
        boundary_mode[i] = 1;
      for ( i = nnodes_all-nnodes_x; i < nnodes_all; i++)
        boundary_mode[i] = 2;
      for (j=0; j < nnodes_y; j++)
        {
          boundary_mode[j*nnodes_x] = 3;
          boundary_mode[(j+1)*nnodes_x-1] = 4;
        }
      boundary_mode[0] = 5;
      boundary_mode[nnodes_x-1] = 6;
      boundary_mode[nnodes_all-nnodes_x] = 7;
      boundary_mode[nnodes_all-1] = 8;
      break;
    case 5:  /*  periodic boundaries  -  interpolation LGLB model  */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 10;
      break;
    case 6:  /*  horizontal walls  -  finite difference model  */
      for ( i = 0; i< nnodes_x; i++)
        boundary_mode[i] = 11;
      for ( i = nnodes_x; i < nnodes_all-nnodes_x; i++)
        boundary_mode[i] = 10;
      for ( i = nnodes_all-nnodes_x; i < nnodes_all; i++)
        boundary_mode[i] = 12;
      break;
    case 7:  /*  vertical walls  -  interpolation LGLB model  */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 10;
      for (j=0; j < nnodes_y; j++)
        {
          boundary_mode[j*nnodes_x] = 13;
          boundary_mode[(j+1)*nnodes_x-1] = 14;
        }
      break;
    case 8:  /*  container  -  interpolation LGLB model  */
      for ( i = 0; i< nnodes_all; i++)
        boundary_mode[i] = 10;
```

```
      for ( i = 0; i< nnodes_x; i++)
        boundary_mode[i] = 11;
      for ( i = nnodes_all-nnodes_x; i < nnodes_all; i++)
        boundary_mode[i] = 12;
      for (j=0; j < nnodes_y; j++)
        {
          boundary_mode[j*nnodes_x] = 13;
          boundary_mode[(j+1)*nnodes_x-1] = 14;
        }
      boundary_mode[0] = 15;
      boundary_mode[nnodes_x-1] = 16;
      boundary_mode[nnodes_all-nnodes_x] = 17;
      boundary_mode[nnodes_all-1] = 18;
      break;
    }
}
```

```c
/***********************************************************************\
*                                                                       *
*  dif9aux.c                                                            *
*                                                                       *
\***********************************************************************/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>

#include "dif9head.h"

void init_lattice_functions(void)
{
  f10 = (double*) malloc(nnodes_all*sizeof(double));
  f11 = (double*) malloc(nnodes_all*sizeof(double));
  f12 = (double*) malloc(nnodes_all*sizeof(double));
  f13 = (double*) malloc(nnodes_all*sizeof(double));
  f14 = (double*) malloc(nnodes_all*sizeof(double));
  f15 = (double*) malloc(nnodes_all*sizeof(double));
  f16 = (double*) malloc(nnodes_all*sizeof(double));
  f17 = (double*) malloc(nnodes_all*sizeof(double));
  f18 = (double*) malloc(nnodes_all*sizeof(double));
  f20 = (double*) malloc(nnodes_all*sizeof(double));
  f21 = (double*) malloc(nnodes_all*sizeof(double));
  f22 = (double*) malloc(nnodes_all*sizeof(double));
  f23 = (double*) malloc(nnodes_all*sizeof(double));
  f24 = (double*) malloc(nnodes_all*sizeof(double));
  f25 = (double*) malloc(nnodes_all*sizeof(double));
  f26 = (double*) malloc(nnodes_all*sizeof(double));
  f27 = (double*) malloc(nnodes_all*sizeof(double));
  f28 = (double*) malloc(nnodes_all*sizeof(double));
  ff10 = (double*) malloc(nnodes_all*sizeof(double));
  ff11 = (double*) malloc(nnodes_all*sizeof(double));
  ff12 = (double*) malloc(nnodes_all*sizeof(double));
  ff13 = (double*) malloc(nnodes_all*sizeof(double));
  ff14 = (double*) malloc(nnodes_all*sizeof(double));
  ff15 = (double*) malloc(nnodes_all*sizeof(double));
  ff16 = (double*) malloc(nnodes_all*sizeof(double));
  ff17 = (double*) malloc(nnodes_all*sizeof(double));
  ff18 = (double*) malloc(nnodes_all*sizeof(double));
  ff20 = (double*) malloc(nnodes_all*sizeof(double));
  ff21 = (double*) malloc(nnodes_all*sizeof(double));
  ff22 = (double*) malloc(nnodes_all*sizeof(double));
  ff23 = (double*) malloc(nnodes_all*sizeof(double));
  ff24 = (double*) malloc(nnodes_all*sizeof(double));
  ff25 = (double*) malloc(nnodes_all*sizeof(double));
  ff26 = (double*) malloc(nnodes_all*sizeof(double));
  ff27 = (double*) malloc(nnodes_all*sizeof(double));
  ff28 = (double*) malloc(nnodes_all*sizeof(double));
  sf10 = (double*) malloc(nnodes_all*sizeof(double));
  sf11 = (double*) malloc(nnodes_all*sizeof(double));
  sf12 = (double*) malloc(nnodes_all*sizeof(double));
  sf13 = (double*) malloc(nnodes_all*sizeof(double));
  sf14 = (double*) malloc(nnodes_all*sizeof(double));
  sf15 = (double*) malloc(nnodes_all*sizeof(double));
  sf16 = (double*) malloc(nnodes_all*sizeof(double));
  sf17 = (double*) malloc(nnodes_all*sizeof(double));
  sf18 = (double*) malloc(nnodes_all*sizeof(double));
  sf20 = (double*) malloc(nnodes_all*sizeof(double));
  sf21 = (double*) malloc(nnodes_all*sizeof(double));
  sf22 = (double*) malloc(nnodes_all*sizeof(double));
  sf23 = (double*) malloc(nnodes_all*sizeof(double));
  sf24 = (double*) malloc(nnodes_all*sizeof(double));
  sf25 = (double*) malloc(nnodes_all*sizeof(double));
  sf26 = (double*) malloc(nnodes_all*sizeof(double));
  sf27 = (double*) malloc(nnodes_all*sizeof(double));
  sf28 = (double*) malloc(nnodes_all*sizeof(double));
  neq10 = (double*) malloc(nnodes_all*sizeof(double));
```



```c
  neq11 = (double*) malloc(nnodes_all*sizeof(double));
  neq12 = (double*) malloc(nnodes_all*sizeof(double));
  neq13 = (double*) malloc(nnodes_all*sizeof(double));
  neq14 = (double*) malloc(nnodes_all*sizeof(double));
  neq15 = (double*) malloc(nnodes_all*sizeof(double));
  neq16 = (double*) malloc(nnodes_all*sizeof(double));
  neq17 = (double*) malloc(nnodes_all*sizeof(double));
  neq18 = (double*) malloc(nnodes_all*sizeof(double));
  neq20 = (double*) malloc(nnodes_all*sizeof(double));
  neq21 = (double*) malloc(nnodes_all*sizeof(double));
  neq22 = (double*) malloc(nnodes_all*sizeof(double));
  neq23 = (double*) malloc(nnodes_all*sizeof(double));
  neq24 = (double*) malloc(nnodes_all*sizeof(double));
  neq25 = (double*) malloc(nnodes_all*sizeof(double));
  neq26 = (double*) malloc(nnodes_all*sizeof(double));
  neq27 = (double*) malloc(nnodes_all*sizeof(double));
  neq28 = (double*) malloc(nnodes_all*sizeof(double));
  uxloc = (double*) malloc(nnodes_all*sizeof(double));
  uyloc = (double*) malloc(nnodes_all*sizeof(double));
  uxloc1 = (double*) malloc(nnodes_all*sizeof(double));
  uyloc1 = (double*) malloc(nnodes_all*sizeof(double));
  uxloc2 = (double*) malloc(nnodes_all*sizeof(double));
  uyloc2 = (double*) malloc(nnodes_all*sizeof(double));
  nv1 = (int*) malloc(nnodes_all*sizeof(int));
  nv2 = (int*) malloc(nnodes_all*sizeof(int));
  nv3 = (int*) malloc(nnodes_all*sizeof(int));
  nv4 = (int*) malloc(nnodes_all*sizeof(int));
  nv5 = (int*) malloc(nnodes_all*sizeof(int));
  nv6 = (int*) malloc(nnodes_all*sizeof(int));
  nv7 = (int*) malloc(nnodes_all*sizeof(int));
  nv8 = (int*) malloc(nnodes_all*sizeof(int));
  boundary_mode = (int*) malloc(nnodes_all*sizeof(int));
  nloc1 = (double*) malloc(nnodes_all*sizeof(double));
  nloc2 = (double*) malloc(nnodes_all*sizeof(double));
}

void free_lattice_functions(void)
{
  free(f10);
  free(f11);
  free(f12);
  free(f13);
  free(f14);
  free(f15);
  free(f16);
  free(f17);
  free(f18);
  free(f20);
  free(f21);
  free(f22);
  free(f23);
  free(f24);
  free(f25);
  free(f26);
  free(f27);
  free(f28);
  free(ff10);
  free(ff11);
  free(ff12);
  free(ff13);
  free(ff14);
  free(ff15);
  free(ff16);
  free(ff17);
  free(ff18);
  free(ff20);
  free(ff21);
  free(ff22);
  free(ff23);
```

```
  free(ff24);
  free(ff25);
  free(ff26);
  free(ff27);
  free(ff28);
  free(sf10);
  free(sf11);
  free(sf12);
  free(sf13);
  free(sf14);
  free(sf15);
  free(sf16);
  free(sf17);
  free(sf18);
  free(sf20);
  free(sf21);
  free(sf22);
  free(sf23);
  free(sf24);
  free(sf25);
  free(sf26);
  free(sf27);
  free(sf28);
  free(neq10);
  free(neq11);
  free(neq12);
  free(neq13);
  free(neq14);
  free(neq15);
  free(neq16);
  free(neq17);
  free(neq18);
  free(neq20);
  free(neq21);
  free(neq22);
  free(neq23);
  free(neq24);
  free(neq25);
  free(neq26);
  free(neq27);
  free(neq28);
  free(uxloc);
  free(uyloc);
  free(uxloc1);
  free(uyloc1);
  free(uxloc2);
  free(uyloc2);
  free(nv1);
  free(nv2);
  free(nv3);
  free(nv4);
  free(nv5);
  free(nv6);
  free(nv7);
  free(nv8);
  free(boundary_mode);
  free(nloc1);
  free(nloc2);
}


void test_distribution_functions(double nf0[], double nf1[], double nf2[],
                                 double nf3[], double nf4[], double nf5[],
                                 double nf6[], double nf7[], double nf8[])
{
  FILE *frez;
  int k;
  frez = fopen(rez_name,"aw");
  for(k=0; k<nnodes_all; k++)
```



```
    {
      if(nf0[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf0=%g\n",
                  iter,k,nf0[k]);
          exit(1);
        }
      if(nf1[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf1=%g\n",
                  iter,k,nf1[k]);
          exit(1);
        }
      if(nf2[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf2=%g\n",
                  iter,k,nf2[k]);
          exit(1);
        }
      if(nf3[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf3=%g\n",
                  iter,k,nf3[k]);
          exit(1);
        }
      if(nf4[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf4=%g\n",
                  iter,k,nf4[k]);
          exit(1);
        }
      if(nf5[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf5=%g\n",
                  iter,k,nf5[k]);
          exit(1);
        }
      if(nf6[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf6=%g\n",
                  iter,k,nf6[k]);
          exit(1);
        }
      if(nf7[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf7=%g\n",
                  iter,k,nf7[k]);
          exit(1);
        }
      if(nf8[k] < 0.00)
        {
          fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf8=%g\n",
                  iter,k,nf8[k]);
          exit(1);
        }
    }
  fclose(frez);
}

void store_distribution_functions(void)
{
  int k;
  for(k=0; k<nnodes_all; k++)
    {
      sf10[k] = f10[k];
      sf11[k] = f11[k];
      sf12[k] = f12[k];
      sf13[k] = f13[k];
      sf14[k] = f14[k];
```

```
        sf15[k] = f15[k];
        sf16[k] = f16[k];
        sf17[k] = f17[k];
        sf18[k] = f18[k];
        sf20[k] = f20[k];
        sf21[k] = f21[k];
        sf22[k] = f22[k];
        sf23[k] = f23[k];
        sf24[k] = f24[k];
        sf25[k] = f25[k];
        sf26[k] = f26[k];
        sf27[k] = f27[k];
        sf28[k] = f28[k];
      }
}
```

```c
/**************************************************************\
 *                                                            *
 *  dif9draw.c                                                *
 *                                                            *
\**************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "dif9head.h"

/* you may modify gray_levels */
  int gray_levels = 63, base = 35;

void xv_new(double n0[],
            double n1[], double n2[], double n3[], double n4[],
            double n5[], double n6[], double n7[], double n8[],
            char arg_name[], double maxval)
{
  int i, val;
  char xv_name[128];
  FILE *fxv;

  sprintf(xv_name, "%s.%06d.xpm", arg_name, iter);
  fxv = fopen(xv_name, "w");
  fprintf(fxv, "/* XPM */\nstatic char * init_xpm[] = {\n");
  fprintf(fxv, "\"%d %d %d %d", nnodes_x, nnodes_y, gray_levels + 1, 1);
  for(i = 0; i <= gray_levels; i++)
    fprintf(fxv, "\",\n\"%c\tg gray%d", base + i,
            (int)(100.00 * i / gray_levels));

                                        /* lattice -> xpm */
  for(i = 0; i <= nnodes_all; i++)
    {
      if(1 == ((i+1) % nnodes_x))
        fprintf(fxv,"\",\n\"");
      val = floor( (n0[i] + n1[i] + n2[i] + n3[i] + n4[i] + n5[i] +
                    n6[i] + n7[i] + n8[i]) * gray_levels / maxval);
      if(val > gray_levels)
        val = gray_levels;
      if(val < 0)
        val = 0;
      fprintf(fxv,"%c", base + val);
    }
  fprintf(fxv,"\"};\n");
  fclose(fxv);
}


void xv(double n0[],
        double n1[], double n2[], double n3[], double n4[],
        double n5[], double n6[], double n7[], double n8[],
        char arg_name[])
{
  FILE *fxv;
  char xv_name[128];
  int i,k,val;
  sprintf(xv_name,"%s.%05d",arg_name,iter);
  fxv = fopen(xv_name,"wt");
  fprintf(fxv,"P2\n%3d%4d\n63\n",nnodes_x,nnodes_y);
  i=0;
  for(k=0; k<nnodes_all; k++)
    {
      i++;
      val = floor((n0[k]+n1[k]+n2[k]+n3[k]+n4[k]+n5[k]+
                   n6[k]+n7[k]+n8[k])*63.0);
      if(val > 63)
```



```c
        val = 63;
      if(val < 0)
        val = 0;
/*      val = 63 - val;   */
      fprintf(fxv,"%3d",val);
      if(i == 15)
        {
          i=0;
          fprintf(fxv,"\n");
        }
    }
  if(i)
    fprintf(fxv,"\n");
  fclose(fxv);
/*
  sprintf(xv_name,"%s.%05dP",arg_name,iter);
  fxv = fopen(xv_name,"wt");
  fprintf(fxv,"P2\n%3d%4d\n63\n",nnlun,nnlat);
  i=0;
  for(k=1; k<=nnod; k++)
    {
      i++;
      val = ( floor((f11[k]*ecx[1]+f12[k]*ecx[2]+f13[k]*ecx[3]+
                     f14[k]*ecx[4]+f15[k]*ecx[5]+f16[k]*ecx[6]+
                     f17[k]*ecx[7]+f18[k]*ecx[8])*1000.));

      if(val > 63)
        val = 63;
      if(val < 0)
        val = 0;
      val = 63 - val;

      fprintf(fxv,"%3d",val);
      if(i == 15)
        {
          i=0;
          fprintf(fxv,"\n");
        }
    }
  if(i)
    fprintf(fxv,"\n");
  fclose(fxv);
*/
}

void dif9_profile(void)
{
  FILE *pntot, *prhotot, *prho1, *prho2, *pomega1, *pomegan1,
    *pomega2, *pomegan2, *pj2, *pu1, *pu2, *pu, *pu2bis, *pubis;

  char ntot_profile_name[128];
  char rhotot_profile_name[128];
  char rho1_profile_name[128];
  char rho2_profile_name[128];
  char omega1_profile_name[128];
  char omegan1_profile_name[128];
  char omega2_profile_name[128];
  char omegan2_profile_name[128];
  char j2_profile_name[128];
  char u1_profile_name[128];
  char u2_profile_name[128];
  char u_profile_name[128];
  char u2bis_profile_name[128];
  char ubis_profile_name[128];

  int i,j,k,kk,nval;
  double nloc1, nloc2, rholoc1, rholoc2, rholoc, omega1loc, omegan1loc,
    omega2loc, omegan2loc, uloc1, uloc2, jloc2, uloc;
  double uloc1bis, uloc2bis, ulocbis;
```

```c
  double rho1, rho2, rho, omega1, omegan1, omega2, omegan2,
    u1, u2, j2, u, u2bis, ubis;
  double ntot, rhotot;

  sprintf(ntot_profile_name,"PNTOTAL%s.%06d",id_name,iter);
  sprintf(rhotot_profile_name,"PRHOTOT%s.%06d",id_name,iter);
  sprintf(rho1_profile_name,"PRHO1%s.%06d",id_name,iter);
  sprintf(rho2_profile_name,"PRHO2%s.%06d",id_name,iter);
  sprintf(omega1_profile_name,"POMEGAMASS1%s.%06d",id_name,iter);
  sprintf(omegan1_profile_name,"POMEGAN1%s.%06d",id_name,iter);
  sprintf(omega2_profile_name,"POMEGAMASS2%s.%06d",id_name,iter);
  sprintf(omegan2_profile_name,"POMEGAN2%s.%06d",id_name,iter);
  sprintf(j2_profile_name,"PJ2%s.%06d",id_name,iter);
  sprintf(u1_profile_name,"PU1%s.%06d",id_name,iter);
  sprintf(u2_profile_name,"PU2%s.%06d",id_name,iter);
  sprintf(u_profile_name,"PU%s.%06d",id_name,iter);
  sprintf(u2bis_profile_name,"PU2BIS%s.%06d",id_name,iter);
  sprintf(ubis_profile_name,"PUBIS%s.%06d",id_name,iter);

  pntot = fopen(ntot_profile_name,"wt");
  prhotot = fopen(rhotot_profile_name,"wt");
  prho1 = fopen(rho1_profile_name,"wt");
  prho2 = fopen(rho2_profile_name,"wt");
  pomega1 = fopen(omega1_profile_name,"wt");
  pomegan1 = fopen(omegan1_profile_name,"wt");
  pomega2 = fopen(omega2_profile_name,"wt");
  pomegan2 = fopen(omegan2_profile_name,"wt");
  pj2 = fopen(j2_profile_name,"wt");
  pu1 = fopen(u1_profile_name,"wt");
  pu2 = fopen(u2_profile_name,"wt");
  pu = fopen(u_profile_name,"wt");
  /*
  pu2bis = fopen(u2bis_profile_name,"wt");
  pubis = fopen(ubis_profile_name,"wt");
  */
  for(i=0; i<nnodes_x; i++)
    {
      ntot = 0.0000;
      rhotot = 0.0000;
      rho1 = 0.0000;
      rho2 = 0.0000;
      omega1 = 0.0000;
      omegan1 = 0.0000;
      omega2 = 0.0000;
      omegan2 = 0.0000;
      j2 = 0.0000;
      u1 = 0.0000;
      u2 = 0.0000;
      u = 0.0000;
      u2bis = 0.0000;
      ubis = 0.0000;
      for(j=0; j<nnodes_y; j++)
        {
          k = i + j * nnodes_x;
          nloc1 = f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+
                  f17[k]+f18[k];
          nloc2 = f20[k]+f21[k]+f22[k]+f23[k]+f24[k]+f25[k]+f26[k]+
                  f27[k]+f28[k];
          /*
          printf("k=%d  nloc1=%g  nloc2=%g\n",k,nloc1,nloc2);
          */
          rholoc1 = mass1 * nloc1;
          rholoc2 = mass2 * nloc2;
          rholoc = rholoc1 + rholoc2;
          /*
          printf("k=%d  rholoc1=%g  rholoc2=%g  rholoc=%g\n",
                 k,rholoc1,rholoc2,rholoc);
          */
          if(rholoc)
```



```c
            {
            omega1loc = rholoc1 / rholoc;
            omega2loc = rholoc2 / rholoc;
            }
          else
            {
              omega1loc = 0.0000;
              omega2loc = 0.0000;
              /*
              printf("k=%d  rholoc=%g\n",k,rholoc);
              */
            }

          if(nloc1+nloc2)
            {
            omegan1loc = nloc1 / (nloc1+nloc2);
            omegan2loc = nloc2 / (nloc1+nloc2);
            }
          else
            {
              omegan1loc = 0.0000;
              omegan2loc = 0.0000;
            }

          if(nloc1)
            uloc1 = (f11[k]*ecx1[1]+f12[k]*ecx1[2]+f13[k]*ecx1[3]+
                     f14[k]*ecx1[4]+f15[k]*ecx1[5]+f16[k]*ecx1[6]+
                     f17[k]*ecx1[7]+f18[k]*ecx1[8]) / nloc1;
          else
            uloc1 = 0.0000;

          if(nloc2)
            uloc2 = (f21[k]*ecx2[1]+f22[k]*ecx2[2]+f23[k]*ecx2[3]+
                     f24[k]*ecx2[4]+f25[k]*ecx2[5]+f26[k]*ecx2[6]+
                     f27[k]*ecx2[7]+f28[k]*ecx2[8]) / nloc2;
          else
            uloc2 = 0.0000;

          /*
          printf("k=%d  uloc1=%g  uloc2=%g\n",k,uloc1,uloc2);
          */

          if(rholoc)
            {
              uloc = ( rholoc1*uloc1 + rholoc2*uloc2 ) / rholoc;
              jloc2 = rholoc2 * (uloc2 - uloc);
            }
          else
            {
              uloc = 0.0000;
              /*
              printf("k=%d  rholoc=%g\n",k,rholoc);
              */
            }

          /*
          if(nloc1)
            uloc1bis = (uloc1 * (1.000 - 0.500*delta_t/tau1) +
                        uloc*0.500*delta_t/tau1)/nloc1;
          else
            uloc1bis = 0.000;
          if(nloc2)
            uloc2bis = (uloc2 * (1.000 - 0.500*delta_t/tau2) +
                        uloc*0.500*delta_t/tau2)/nloc2;
          else
            uloc2bis = 0.000;
          ulocbis = (rholoc1*uloc1*(1.000 - 0.500*delta_t/tau1) +
                     rholoc2*uloc2*(1.000 - 0.500*delta_t/tau2) +
                     rholoc1*uloc*0.500*delta_t/tau1 +
```

```
                   rholoc2*uloc*0.500*delta_t/tau2) / rholoc;
          */

          ntot += (nloc1 + nloc2);
          rhotot += (rholoc1 + rholoc2);
          rho1 += rholoc1;
          rho2 += rholoc2;
          omega1 += omega1loc;
          omega2 += omega2loc;
          omegan1 += omegan1loc;
          omegan2 += omegan2loc;
          j2 += jloc2;
          u1 += uloc1;
          u2 += uloc2;
          u += uloc;
          /*
          u2bis += uloc2bis;
          ubis += ulocbis;
          */
        }
      ntot /= ((double) nnodes_y);
      rhotot /= ((double) nnodes_y);
      rho1 /= ((double) nnodes_y);
      rho2 /= ((double) nnodes_y);
      omega1 /= ((double) nnodes_y);
      omega2 /= ((double) nnodes_y);
      omegan1 /= ((double) nnodes_y);
      omegan2 /= ((double) nnodes_y);
      j2 /= ((double) nnodes_y);
      u1 /= ((double) nnodes_y);
      u2 /= ((double) nnodes_y);
      u /= ((double) nnodes_y);
      u2bis /= ((double) nnodes_y);
      ubis /= ((double) nnodes_y);
      fprintf(pntot,"%g\n",ntot);
      fprintf(prhotot,"%g\n",rhotot);
      fprintf(prho1,"%g\n",rho1);
      fprintf(prho2,"%g\n",rho2);
      fprintf(pomega1,"%lf\n",omega1);
      fprintf(pomegan1,"%lf\n",omegan1);
      fprintf(pomega2,"%lf\n",omega2);
      fprintf(pomegan2,"%lf\n",omegan2);
      fprintf(pj2,"%lf\n",j2);
      fprintf(pu1,"%lf\n",u1);
      fprintf(pu2,"%lf\n",u2);
      fprintf(pu,"%lf\n",u);
      /*
      fprintf(pu2bis,"%lf\n",u2bis);
      fprintf(pubis,"%lf\n",ubis);
      */
    }

  fclose(pntot);
  fclose(prhotot);
  fclose(prho1);
  fclose(prho2);
  fclose(pomega1);
  fclose(pomegan1);
  fclose(pomega2);
  fclose(pomegan2);
  fclose(pj2);
  fclose(pu1);
  fclose(pu2);
  fclose(pu);
  /*
  fclose(pu2bis);
  fclose(pubis);
  */
}
```



```
void test_ntot(void)
{
  FILE *ptot;
  int k;
  double ntot=0.0000;
  char ntot_name[128];
  sprintf(ntot_name,"PNTOT%s",id_name);
  ptot = fopen(ntot_name,"aw");
  for(k=0; k<nnodes_all; k++)
    {
      ntot += f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+f17[k]+f18[k];
    }
  fprintf(ptot,"%d  %25.20lf\n",iter,ntot);
  fclose(ptot);
}

void quiver(void)
{
  FILE *pux, *puy, *pxc, *pyc, *pxn, *pyn;
  double nloc1, nloc2;
  double ux1, uy1, ux2, uy2, ux, uy, xc, yc;
  int i,j,k,ipas=2,jpas=2, xn, yn;
  char ux_name[128], uy_name[128];
  char cx_name[128], cy_name[128];
  char nx_name[128], ny_name[128];
  sprintf(ux_name,"PUX%s.%06d",id_name,iter);
  sprintf(uy_name,"PUY%s.%06d",id_name,iter);
  sprintf(cx_name,"PCX%s.%06d",id_name,iter);
  sprintf(cy_name,"PCY%s.%06d",id_name,iter);
  sprintf(nx_name,"PNX%s.%06d",id_name,iter);
  sprintf(ny_name,"PNY%s.%06d",id_name,iter);
  pux = fopen(ux_name,"w");
  puy = fopen(uy_name,"w");
  pxc = fopen(cx_name,"w");
  pyc = fopen(cy_name,"w");
  pxn = fopen(nx_name,"w");
  pyn = fopen(ny_name,"w");
  for(j=0; j<nnodes_y; j+=jpas)
    {
      k = j*nnodes_x;
      for(i=0; i<nnodes_x; i+=ipas)
        {
          nloc1 = f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+
                  f17[k]+f18[k];
          nloc2 = f20[k]+f21[k]+f22[k]+f23[k]+f24[k]+f25[k]+f26[k]+
                  f17[k]+f18[k];
          if(nloc1)
            {
              ux1 = (f11[k]*ecx[1]+f12[k]*ecx[2]+f13[k]*ecx[3]+
                     f14[k]*ecx[4]+f15[k]*ecx[5]+f16[k]*ecx[6]+
                     f17[k]*ecx[7]+f18[k]*ecx[8]) / nloc1;
              uy1 = (f11[k]*ecy[1]+f12[k]*ecy[2]+f13[k]*ecy[3]+
                     f14[k]*ecy[4]+f15[k]*ecy[5]+f16[k]*ecy[6]+
                     f17[k]*ecy[7]+f18[k]*ecy[8]) / nloc1;
            }
          else
            {
              ux1 = 0.0000;
              uy1 = 0.0000;
            }
          if(nloc2)
            {
              ux2 = (f21[k]*ecx[1]+f22[k]*ecx[2]+f23[k]*ecx[3]+
                     f24[k]*ecx[4]+f25[k]*ecx[5]+f26[k]*ecx[6]+
                     f27[k]*ecx[7]+f28[k]*ecx[8]) / nloc2;
              uy2 = (f21[k]*ecy[1]+f22[k]*ecy[2]+f23[k]*ecy[3]+
                     f24[k]*ecy[4]+f25[k]*ecy[5]+f26[k]*ecy[6]+
                     f27[k]*ecy[7]+f28[k]*ecy[8]) / nloc2;
```

```
          }
        else
          {
            ux2 = 0.0000;
            uy2 = 0.0000;
          }
        ux = (mass1 * nloc1 * ux1 + mass2 * nloc2 * ux2) /
             (mass1 * nloc1 + mass2 * nloc2);
        uy = (mass1 * nloc1 * uy1 + mass2 * nloc2 * uy2) /
             (mass1 * nloc1 + mass2 * nloc2);
        xc = delta_x * ((double) i);
        yc = delta_y * ((double) j);
        xn = i;
        yn = j;
        fprintf(pux,"  %lf",ux);
        fprintf(puy,"  %lf",uy);
        fprintf(pxc,"  %lf",xc);
        fprintf(pyc,"  %lf",yc);
        fprintf(pxn,"  %5d",xn);
        fprintf(pyn,"  %5d",yn);
        k+= ipas;
      }
    fprintf(pux,"\n");
    fprintf(puy,"\n");
    fprintf(pxc,"\n");
    fprintf(pyc,"\n");
    fprintf(pxn,"\n");
    fprintf(pyn,"\n");
  }
  fclose(pux);
  fclose(puy);
  fclose(pxc);
  fclose(pyc);
  fclose(pxn);
  fclose(pyn);
}
```

```c
/*********************************************************************\
*                                                                     *
*  dif9fourier.c  --  fft                                             *
*                                                                     *
\*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "dif9head.h"

void four1(double data[2*nnodes_x], int nn, int isign)
{
  int i, istep, j, m, mmax, n;
  double tempi, tempr;
  double theta, wi, wpi, wpr, wr, wtemp;
  n = 2 * nn;
  j = 1;
  for(i=1; i<=n; i+=2)
    {
      if(j>i)
        {
          tempr = data[j-1];
          tempi = data[j];
          data[j-1] = data[i-1];
          data[j] = data[i];
          data[i-1] = tempr;
          data[i] = tempi;
        }
      m = n/2;
      while( (m>=2) && (j>= m) )
        {
          j = j-m;
          m /= 2;
        }
      j += m;
    }
  mmax = 2;
  while(n > mmax)
    {
      istep = 2 * mmax;
      theta = 6.28318530717959 / (isign * mmax);
      wpr = -2.000 * sin(0.5*theta) * sin(0.5*theta);
      wpi = sin(theta);
      wr = 1.000;
      wi = 0.000;
      for(m=1; m<= mmax; m+=2)
        {
          for(i=m; i<=n; i+=istep)
            {
              j = i+mmax;
              tempr = wr * data[j-1] - wi * data[j];
              tempi = wr * data[j] + wi * data[j-1];
              data[j-1] = data[i-1] - tempi;
              data[j] = data[i] - tempi;
              data[i-1] = data[i-1] + tempr;
              data[i] = data[i] + tempi;
            }
          wtemp = wr;
          wr = wr*wpr - wi*wpi + wr;
          wi = wi*wpr + wtemp*wpi + wi;
        }
      mmax = istep;
    }
}

void realft(double data[nnodes_x], int n, int isign)
```



```c
{
  int i, i1, i2, i3, i4, n2p3;
  double c1, c2, h1i, h1r, h2i, h2r, wis, wrs;
  double theta, wi, wpi, wpr, wr, wtemp;
  theta = 3.141592653589793 / (((double) n) / 2.000);
  c1 = 0.5;
  if(isign == 1)
    {
      c2 = -0.5;
      four1(data,n/2,1);
    }
  else
    {
      c2 = 0.5;
      theta = -theta;
    }
  wpr = -2.000 * sin(0.500*theta) * sin(0.500*theta);
  wpi = sin(theta);
  wr = 1.000 + wpr;
  wi = wpi;
  n2p3 = n + 3;
  for(i=2; i<= n/4; i++)
    {
      i1 = 2*i - 1;
      i2 = i1+1;
      i3 = n2p3 - i2;
      i4 = i3 +1;
      wrs = wr;
      wis = wi;
      h1r = c1 * (data[i1-1] + data[i3-1]);
      h1i = c1 * (data[i2-1] - data[i4-1]);
      h2r = -c2 * (data[i2-1] + data[i4-1]);
      h2i = c2 * (data[i1-1] - data[i3-1]);
      data[i1-1] = h1r + wrs*h2r - wis*h2i;
      data[i2-1] = h1i + wrs*h2i + wis*h2r;
      data[i3-1] = h1r - wrs*h2r + wis*h2r;
      data[i4-1] = -h1i + wrs*h2i + wis*h2r;
      wtemp = wr;
      wr = wr*wpr - wi*wpi + wr;
      wi = wi*wpr + wtemp*wpi + wi;
    }
  if(isign == 1)
    {
      h1r = data[0];
      data[0] = h1r + data[1];
      data[1] = h1r - data[1];
    }
  else
    {
      h1r = data[0];
      data[0] = c1*(h1r + data[1]);
      data[1] = c1 * (h1r - data[1]);
      four1(data,n/2,-1);
    }
}
```

```
/************************************************************************\
*                                                                        *
*  dif9main.c                                                            *
*                                                                        *
\************************************************************************/

#define MAIN_HEADER

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "dif9head.h"

void getavec_square(void);
void dif9_input(void);
void dif9_output(void);
void build_names(int);
void test_ntot(void);
void test_distribution_functions(double n0[], double n1[], double n2[],
                                 double n3[], double n4[], double n5[],
                                 double n6[], double n7[], double n8[]);
void xv_new(double n0[], double n1[], double n2[], double n3[],
            double n4[], double n5[], double n6[], double n7[], double n8[],
            char filename[], double maxval);
void xv(double n0[], double n1[], double n2[], double n3[],
        double n4[], double n5[], double n6[], double n7[], double n8[],
        char filename[]);
void init_ecx_nine(void);
void init_arrays_nine_square(void);
void init_arrays_nine_square_aux(void);
void dif9_profile(void);


void compute_local_speeds(double nf10[], double nf11[], double nf12[],
                          double nf13[], double nf14[], double nf15[],
                          double nf16[], double nf17[], double nf18[],
                          double nf20[], double nf21[], double nf22[],
                          double nf23[], double nf24[], double nf25[],
                          double nf26[], double nf27[], double nf28[])
{
  int k;
  double dummy;

  for(k=0; k<nnodes_all; k++)
    {
      nloc1[k] = nf10[k]+nf11[k]+nf12[k]+nf13[k]+nf14[k]+
                 nf15[k]+nf16[k]+nf17[k]+nf18[k];
      nloc2[k] = nf20[k]+nf21[k]+nf22[k]+nf23[k]+nf24[k]+
                 nf25[k]+nf26[k]+nf27[k]+nf28[k];

      if(key_scheme == 10)
        {
          uxloc1[k] = 0.0000000;
          uyloc1[k] = 0.0000000;
          uxloc2[k] = 0.0000000;
          uyloc2[k] = 0.0000000;
          uxloc[k] = 0.00000000;
          uyloc[k] = 0.00000000;
        }
      else
        {
          switch(boundary_mode[k])
            {
            case 0:
              uxloc1[k] = (nf11[k]*ecx1[1]+nf12[k]*ecx1[2]+nf13[k]*ecx1[3]+
                           nf14[k]*ecx1[4]+nf15[k]*ecx1[5]+nf16[k]*ecx1[6]+
                           nf17[k]*ecx1[7]+nf18[k]*ecx1[8]);
              uyloc1[k] = (nf11[k]*ecy1[1]+nf12[k]*ecy1[2]+nf13[k]*ecy1[3]+
```



```
                           nf14[k]*ecy1[4]+nf15[k]*ecy1[5]+nf16[k]*ecy1[6]+
                           nf17[k]*ecy1[7]+nf18[k]*ecy1[8]);

              uxloc2[k] = (nf21[k]*ecx2[1]+nf22[k]*ecx2[2]+nf23[k]*ecx2[3]+
                           nf24[k]*ecx2[4]+nf25[k]*ecx2[5]+nf26[k]*ecx2[6]+
                           nf27[k]*ecx2[7]+nf28[k]*ecx2[8]);
              uyloc2[k] = (nf21[k]*ecy2[1]+nf22[k]*ecy2[2]+nf23[k]*ecy2[3]+
                           nf24[k]*ecy2[4]+nf25[k]*ecy2[5]+nf26[k]*ecy2[6]+
                           nf27[k]*ecy2[7]+nf28[k]*ecy2[8]);

              dummy = (mass1*nloc1[k])/tau1 + (mass2*nloc2[k])/tau2;
              if(dummy)
                {
                  uxloc[k] = ((mass1 * uxloc1[k]) / tau1 +
                              (mass2 * uxloc2[k]) / tau2) / dummy;
                  uyloc[k] = ((mass1 * uyloc1[k]) / tau1 +
                              (mass2 * uyloc2[k]) / tau2) / dummy;
                }
              else
                {
                  uxloc[k] = 0.0000;
                  uyloc[k] = 0.0000;
                }
              if(nloc1[k])
                {
                  uxloc1[k] /= nloc1[k];
                  uyloc1[k] /= nloc1[k];
                }
              else
                {
                  uxloc1[k] = 0.0000;
                  uyloc1[k] = 0.0000;
                }
              if(nloc2[k])
                {
                  uxloc2[k] /= nloc2[k];
                  uyloc2[k] /= nloc2[k];
                }
              else
                {
                  uxloc2[k] = 0.0000;
                  uyloc2[k] = 0.0000;
                }
              /*
                printf("k===%d  %lf  %lf  %lf  %lf  %lf  %lf\n",
                k,uxloc1[k],uyloc1[k],
                uxloc2[k],uyloc2[k],
                uxloc[k],uyloc[k]);
                */
              break;
            default:
              uxloc1[k] = 0.0000000;
              uyloc1[k] = 0.0000000;
              uxloc2[k] = 0.0000000;
              uyloc2[k] = 0.0000000;
              uxloc[k] = 0.00000000;
              uyloc[k] = 0.00000000;
              break;
            }
        }
    }
}

void compute_equilibrium_distributions2(void)
{
  int k;
  double dummy, uu, uu1, uu2, s2;
  double nt1, nt2, net1, net2;
```

```
  s2 = sqrt((double) 2);

  /*
  nt1 = (double) 0;
  nt2 = (double) 0;
  net1 = (double) 0;
  net2 = (double) 0;

  printf("\n====  iter=%d  k=%d  nt1=%g  net1=%g  nt2=%g  net2=%g\n\n",
    iter,k,nt1,net1,nt2,net2);
  */

  for(k=0; k<nnodes_all; k++)
    {
      if(nloc2[k])
        {
          if(nloc1[k])
            {
              uxloc[k] = ( alpha*mass1*nloc1[k]*uxloc1[k] +
                          alpha1*mass2*nloc2[k]*uxloc2[k]) /
                         (mass1*nloc1[k]);
              uyloc[k] = ( alpha*mass1*nloc1[k]*uyloc1[k] +
                          alpha1*mass2*nloc2[k]*uyloc2[k]) /
                         (mass1*nloc1[k]);
            }
          else
            {
              uxloc[k] = uxloc1[k];
              uyloc[k] = uyloc1[k];
            }
        }
      else
        {
          uxloc[k] = uxloc1[k];
          uyloc[k] = uyloc1[k];
        }
      uu = uxloc[k]*uxloc[k]+uyloc[k]*uyloc[k];
      uu1 = three_over_two * uu / cspeed12;
      uu2 = three_over_two * uu / cspeed22;

      neq10[k] = w0*nloc1[k] * (1.000 - uu1);

      dummy = (ecx1[1]*uxloc[k] + ecy1[1]*uyloc[k]) / cspeed12;
      neq11[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[2]*uxloc[k] + ecy1[2]*uyloc[k]) / cspeed12;
      neq12[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[3]*uxloc[k] + ecy1[3]*uyloc[k]) / cspeed12;
      neq13[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[4]*uxloc[k] + ecy1[4]*uyloc[k]) / cspeed12;
      neq14[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[5]*uxloc[k] + ecy1[5]*uyloc[k]) / cspeed12;
      neq15[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[6]*uxloc[k] + ecy1[6]*uyloc[k]) / cspeed12;
      neq16[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[7]*uxloc[k] + ecy1[7]*uyloc[k]) / cspeed12;
      neq17[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[8]*uxloc[k] + ecy1[8]*uyloc[k]) / cspeed12;
      neq18[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);

      if(nloc1[k])
        {
```



```
          if(nloc2[k])
            {
              uxloc[k] = ( alpha*mass2*nloc2[k]*uxloc2[k] +
                          alpha1*mass1*nloc1[k]*uxloc1[k]) /
                         (mass2 * nloc2[k]);
              uyloc[k] = ( alpha*mass2*nloc2[k]*uyloc2[k] +
                          alpha1*mass1*nloc1[k]*uyloc1[k]) /
                         (mass2*nloc2[k]);
            }
          else
            {
              uxloc[k] = uxloc2[k];
              uyloc[k] = uyloc2[k];
            }
        }
      else
        {
          uxloc[k] = uxloc2[k];
          uyloc[k] = uyloc2[k];
        }
      uu = uxloc[k]*uxloc[k]+uyloc[k]*uyloc[k];
      uu1 = three_over_two * uu / cspeed12;
      uu2 = three_over_two * uu / cspeed22;

      neq20[k] = w0*nloc2[k] * (1.000 - uu2);

      dummy = (ecx2[1]*uxloc[k] + ecy2[1]*uyloc[k]) / cspeed22;
      neq21[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[2]*uxloc[k] + ecy2[2]*uyloc[k]) / cspeed22;
      neq22[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[3]*uxloc[k] + ecy2[3]*uyloc[k]) / cspeed22;
      neq23[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[4]*uxloc[k] + ecy2[4]*uyloc[k]) / cspeed22;
      neq24[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[5]*uxloc[k] + ecy2[5]*uyloc[k]) / cspeed22;
      neq25[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[6]*uxloc[k] + ecy2[6]*uyloc[k]) / cspeed22;
      neq26[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[7]*uxloc[k] + ecy2[7]*uyloc[k]) / cspeed22;
      neq27[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[8]*uxloc[k] + ecy2[8]*uyloc[k]) / cspeed22;
      neq28[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);

      /*
      nt1 +=nloc1[k];
      nt2 +=nloc2[k];
      net1+=neq10[k]+neq11[k]+neq12[k]+neq13[k]+neq14[k]+neq15[k]
           +neq16[k]+neq17[k]+neq18[k];
      net2+=neq20[k]+neq21[k]+neq22[k]+neq23[k]+neq24[k]+neq25[k]
           +neq26[k]+neq27[k]+neq28[k];
      if((iter > 99) && (k<300) && (k>200))
        {
      printf("iter=%d  k=%d  nloc1=%g  neq1=%g  nloc2=%g  neq2=%g\n",
             iter,k,nloc1[k],neq10[k]+neq11[k]+
             neq12[k]+neq13[k]+neq14[k]+neq15[k]+neq16[k]+neq17[k]+neq18[k],
             nloc2[k],neq20[k]+neq21[k]+
             neq22[k]+neq23[k]+neq24[k]+neq25[k]+neq26[k]+neq27[k]+neq28[k]);
        }
      */
      /*
      printf("k=%d  uxloc1=%g  uxloc2=%g  uxloc=%g\n",k,uxloc1[k],uxloc2[k],uxloc[k]);
```

```
        printf("k=%d  uyloc1=%g  uyloc2=%g  uyloc=%g\n",k,uyloc1[k],uyloc2[k],uyloc[k]);
        printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
               f10[k],f11[k],f12[k],f13[k],f14[k],f15[k],
               f16[k],f17[k],f18[k]);
        printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
               f20[k],f21[k],f22[k],f23[k],f24[k],f25[k],
               f26[k],f27[k],f28[k]);
        printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
               ff10[k],ff11[k],ff12[k],ff13[k],ff14[k],ff15[k],
               ff16[k],ff17[k],ff18[k]);
        printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
               ff20[k],ff21[k],ff22[k],ff23[k],ff24[k],ff25[k],
               ff26[k],ff27[k],ff28[k]);
        printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
               neq10[k],neq11[k],neq12[k],neq13[k],neq14[k],neq15[k],
               neq16[k],neq17[k],neq18[k]);
        printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
               neq20[k],neq21[k],neq22[k],neq23[k],neq24[k],neq25[k],
               neq26[k],neq27[k],neq28[k]);
        */
      }
/*
  printf("\n====  iter=%d  k=%d  nt1=%g  net1=%g  nt2=%g  net2=%g\n\n",
    iter,k,nt1,net1,nt2,net2);
*/
}

void compute_equilibrium_distributions(void)
{
  int k;
  double dummy, uu, uu1, uu2, s2;
  double nt1, nt2, net1, net2;

  s2 = sqrt((double) 2);

  /*
  nt1 = (double) 0;
  nt2 = (double) 0;
  net1 = (double) 0;
  net2 = (double) 0;

  printf("\n====  iter=%d  k=%d  nt1=%g  net1=%g  nt2=%g  net2=%g\n\n",
    iter,k,nt1,net1,nt2,net2);
  */

  for(k=0; k<nnodes_all; k++)
    {
      uu = uxloc[k]*uxloc[k]+uyloc[k]*uyloc[k];
      uu1 = three_over_two * uu / cspeed12;
      uu2 = three_over_two * uu / cspeed22;

      neq10[k] = w0*nloc1[k] * (1.000 - uu1);

      dummy = (ecx1[1]*uxloc[k] + ecy1[1]*uyloc[k]) / cspeed12;
      neq11[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[2]*uxloc[k] + ecy1[2]*uyloc[k]) / cspeed12;
      neq12[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[3]*uxloc[k] + ecy1[3]*uyloc[k]) / cspeed12;
      neq13[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[4]*uxloc[k] + ecy1[4]*uyloc[k]) / cspeed12;
      neq14[k] = w1*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[5]*uxloc[k] + ecy1[5]*uyloc[k]) / cspeed12;
      neq15[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[6]*uxloc[k] + ecy1[6]*uyloc[k]) / cspeed12;
```



```
      neq16[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[7]*uxloc[k] + ecy1[7]*uyloc[k]) / cspeed12;
      neq17[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);
      dummy = (ecx1[8]*uxloc[k] + ecy1[8]*uyloc[k]) / cspeed12;
      neq18[k] = w2*nloc1[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu1);

      neq20[k] = w0*nloc2[k] * (1.000 - uu2);

      dummy = (ecx2[1]*uxloc[k] + ecy2[1]*uyloc[k]) / cspeed22;
      neq21[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[2]*uxloc[k] + ecy2[2]*uyloc[k]) / cspeed22;
      neq22[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[3]*uxloc[k] + ecy2[3]*uyloc[k]) / cspeed22;
      neq23[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[4]*uxloc[k] + ecy2[4]*uyloc[k]) / cspeed22;
      neq24[k] = w1*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[5]*uxloc[k] + ecy2[5]*uyloc[k]) / cspeed22;
      neq25[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[6]*uxloc[k] + ecy2[6]*uyloc[k]) / cspeed22;
      neq26[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[7]*uxloc[k] + ecy2[7]*uyloc[k]) / cspeed22;
      neq27[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);
      dummy = (ecx2[8]*uxloc[k] + ecy2[8]*uyloc[k]) / cspeed22;
      neq28[k] = w2*nloc2[k] * (1.000 + three * dummy +
                                nine_over_two * dummy * dummy - uu2);

      /*
      nt1 +=nloc1[k];
      nt2 +=nloc2[k];
      net1+=neq10[k]+neq11[k]+neq12[k]+neq13[k]+neq14[k]+neq15[k]
          +neq16[k]+neq17[k]+neq18[k];
      net2+=neq20[k]+neq21[k]+neq22[k]+neq23[k]+neq24[k]+neq25[k]
          +neq26[k]+neq27[k]+neq28[k];
      if((iter > 99) && (k<300) && (k>200))
        {
      printf("iter=%d  k=%d  nloc1=%g  neq1=%g  nloc2=%g  neq2=%g\n",
             iter,k,nloc1[k],neq10[k]+neq11[k]+
             neq12[k]+neq13[k]+neq14[k]+neq15[k]+neq16[k]+neq17[k]+neq18[k],
             nloc2[k],neq20[k]+neq21[k]+
             neq22[k]+neq23[k]+neq24[k]+neq25[k]+neq26[k]+neq27[k]+neq28[k]);
        }
      */
      /*
      printf("k=%d  uxloc1=%g  uxloc2=%g  uxloc=%g\n",k,uxloc1[k],uxloc2[k],uxloc[k]);
      printf("k=%d  uyloc1=%g  uyloc2=%g  uyloc=%g\n",k,uyloc1[k],uyloc2[k],uyloc[k]);
      printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
             f10[k],f11[k],f12[k],f13[k],f14[k],f15[k],
             f16[k],f17[k],f18[k]);
      printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
             f20[k],f21[k],f22[k],f23[k],f24[k],f25[k],
             f26[k],f27[k],f28[k]);
      printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
             ff10[k],ff11[k],ff12[k],ff13[k],ff14[k],ff15[k],
             ff16[k],ff17[k],ff18[k]);
      printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
             ff20[k],ff21[k],ff22[k],ff23[k],ff24[k],ff25[k],
             ff26[k],ff27[k],ff28[k]);
      printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
             neq10[k],neq11[k],neq12[k],neq13[k],neq14[k],neq15[k],
```

```c
          neq16[k],neq17[k],neq18[k]);
      printf("%g  %g  %g  %g  %g  %g  %g  %g  %g\n",
          neq20[k],neq21[k],neq22[k],neq23[k],neq24[k],neq25[k],
          neq26[k],neq27[k],neq28[k]);
      */
    }
/*
 printf("\n==== iter=%d  k=%d  nt1=%g  net1=%g  nt2=%g  net2=%g\n\n",
   iter,k,nt1,net1,nt2,net2);
*/
}

runge_kutta(int key_point,
            double mass, double tau, double cspeed,
            double nf0[], double nf1[], double nf2[], double nf3[],
            double nf4[], double nf5[], double nf6[],
            double nf7[], double nf8[],
            double nff0[], double nff1[], double nff2[], double nff3[],
            double nff4[], double nff5[], double nff6[],
            double nff7[], double nff8[],
            double sf0[], double sf1[], double sf2[], double sf3[],
            double sf4[], double sf5[], double sf6[],
            double sf7[], double sf8[],
            double neq0[], double neq1[], double neq2[], double neq3[],
            double neq4[], double neq5[], double neq6[],
            double neq7[], double neq8[])
{
  int i, j, k;
  double ctau, cgrad, cgradx, cgrady, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  if(key_point)
    {
      ctau = delta_t / tau;
      cgrad = cspeed * delta_t / delta_x;
      cgradx = cspeed * delta_t / delta_x;
      cgrady = cspeed * delta_t / delta_y;
      cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
      cgradx2 = cspeed * delta_t / (2.000*delta_x);
      cgrady2 = cspeed * delta_t / (2.000*delta_y);
      cgradxy2 = cspeed * delta_t / (2.000*
                              sqrt(delta_x*delta_x + delta_y*delta_y));
      dummy_force = (delta_t * cspeed) / (kboltz * temp);
    }
  else
    {
      ctau = delta_t / (2.000 * tau);
      cgrad = cspeed * delta_t / (2.000 * delta_x);
      cgradx = cspeed * delta_t / (2.000 * delta_x);
      cgrady = cspeed * delta_t / (2.000 * delta_y);
      cgradxy=cspeed*delta_t/(2.000*sqrt(delta_x*delta_x+delta_y*delta_y));
      cgradx2 = cspeed * delta_t / (4.000 * delta_x);
      cgrady2 = cspeed * delta_t / (4.000 * delta_y);
      cgradxy2=cspeed*delta_t/(4.000*sqrt(delta_x*delta_x+delta_y*delta_y));
      dummy_force = (delta_t * cspeed) / (2.000 * kboltz * temp);
    }

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */

          /*  first order upwind scheme  */

          nff0[k] = sf0[k] - (nf0[k]-neq0[k])*ctau;
          if(iter < 0)  /*  second order attempt  */
```



```c
            {
             nff1[k] = sf1[k] - (nf1[k]-neq1[k])*ctau -
               cgradx2*ecx[1]*(3.000*nf1[k]-4.000*nf1[nv3[k]]+
                               nf1[nv3[nv3[k]]]);
            }
          else
            {
           nff1[k] = sf1[k] - (nf1[k]-neq1[k])*ctau -
             cgradx*ecx[1]*(nf1[k]-nf1[nv3[k]]);
            }
           nff2[k] = sf2[k] - (nf2[k]-neq2[k])*ctau -
             cgrady*ecy[2]*(nf2[k]-nf2[nv4[k]]);
          if(iter < 0)
            {
             nff3[k] = sf3[k] - (nf3[k]-neq3[k])*ctau -
               cgradx2*ecx[3]*(-3.000*nf3[k]+4.000*nf3[nv1[k]]-
                               -nf3[nv1[nv1[k]]]);
            }
          else
            {
           nff3[k] = sf3[k] - (nf3[k]-neq3[k])*ctau -
             cgradx*ecx[3]*(nf3[nv1[k]]-nf3[k]);
            }
           nff4[k] = sf4[k] - (nf4[k]-neq4[k])*ctau -
             cgrady*ecy[4]*(nf4[nv2[k]]-nf4[k]);
           nff5[k] = sf5[k] - (nf5[k]-neq5[k])*ctau -
             cgradx*ecx[5]*(nf5[k]-nf5[nv3[k]]) -
             cgrady*ecy[5]*(nf5[k]-nf5[nv4[k]]);
           nff6[k] = sf6[k] - (nf6[k]-neq6[k])*ctau -
             cgradx*ecx[6]*(nf6[nv1[k]]-nf6[k]) -
             cgrady*ecy[6]*(nf6[k]-nf6[nv4[k]]);
           nff7[k] = sf7[k] - (nf7[k]-neq7[k])*ctau -
             cgradx*ecx[7]*(nf7[nv1[k]]-nf7[k]) -
             cgrady*ecy[7]*(nf7[nv2[k]]-nf7[k]);
           nff8[k] = sf8[k] - (nf8[k]-neq8[k])*ctau -
             cgradx*ecx[8]*(nf8[k]-nf8[nv3[k]]) -
             cgrady*ecy[8]*(nf8[nv2[k]]-nf8[k]);

           if(key_force)
             {
              prodscal = force_x * uxloc[k] + force_y * uyloc[k];
              nff0[k] += dummy_force *
                (force_x*ecx[0]+force_y*ecy[0]-prodscal)*neq0[k];
              nff1[k] += dummy_force *
                (force_x*ecx[1]+force_y*ecy[1]-prodscal)*neq1[k];
              nff2[k] += dummy_force *
                (force_x*ecx[2]+force_y*ecy[2]-prodscal)*neq2[k];
              nff3[k] += dummy_force *
                (force_x*ecx[3]+force_y*ecy[3]-prodscal)*neq3[k];
              nff4[k] += dummy_force *
                (force_x*ecx[4]+force_y*ecy[4]-prodscal)*neq4[k];
              nff5[k] += dummy_force *
                (force_x*ecx[5]+force_y*ecy[5]-prodscal)*neq5[k];
              nff6[k] += dummy_force *
                (force_x*ecx[6]+force_y*ecy[6]-prodscal)*neq6[k];
             }
           break;

         case 1:  /*  bottom wall  */

           break;

         case 2:   /*  top wall  */

           break;

         case 3:   /*  left wall  */

           nff0[k] = sf0[k] - (nf0[k]-neq0[k])*ctau;
```

```c
        nff1[k] = sf1[k] - (nf1[k]-neq1[k])*ctau -
          cgradx*ecx[1]*(nf1[k]-nf3[k]);
        nff2[k] = sf2[k] - (nf2[k]-neq2[k])*ctau -
          cgrady*ecy[2]*(nf2[k]-nf2[nv4[k]]);
        nff3[k] = sf3[k] - (nf3[k]-neq3[k])*ctau -
          cgradx*ecx[3]*(nf3[nv1[k]]-nf3[k]);
        nff4[k] = sf4[k] - (nf4[k]-neq4[k])*ctau -
          cgrady*ecy[4]*(nf4[nv2[k]]-nf4[k]);
        nff5[k] = sf5[k] - (nf5[k]-neq5[k])*ctau -
          cgradx*ecx[5]*(nf5[k]-nf5[k]) -
          cgrady*ecy[5]*(nf5[k]-nf5[nv4[k]]);
        nff6[k] = sf6[k] - (nf6[k]-neq6[k])*ctau -
          cgradx*ecx[6]*(nf6[nv1[k]]-nf6[k]) -
          cgrady*ecy[6]*(nf6[k]-nf6[nv4[k]]);
        nff7[k] = sf7[k] - (nf7[k]-neq7[k])*ctau -
          cgradx*ecx[7]*(nf7[nv1[k]]-nf7[k]) -
          cgrady*ecy[7]*(nf7[nv2[k]]-nf7[k]);
        nff8[k] = sf8[k] - (nf8[k]-neq8[k])*ctau -
          cgradx*ecx[8]*(nf8[k]-nf8[k]) -
          cgrady*ecy[8]*(nf8[nv2[k]]-nf8[k]);

        /*
        nff5[k] = sf5[k] - (nf5[k]-neq5[k])*ctau -
          cgradxy*edxy*(nf5[k]-nf7[k]);
        nff6[k] = sf6[k] - (nf6[k]-neq6[k])*ctau -
          cgradxy*edxy*(nf6[k]-nf6[nv8[k]]);
        nff7[k] = sf7[k] - (nf7[k]-neq7[k])*ctau -
          cgradxy*edxy*(nf7[k]-nf7[nv5[k]]);
        nff8[k] = sf8[k] - (nf8[k]-neq8[k])*ctau -
          cgradxy*edxy*(nf8[k]-nf6[k]);
        */
        break;

      case 4:   /*  right wall  */

        nff0[k] = sf0[k] - (nf0[k]-neq0[k])*ctau;
        nff1[k] = sf1[k] - (nf1[k]-neq1[k])*ctau -
          cgradx*ecx[1]*(nf1[k]-nf1[nv3[k]]);
        nff2[k] = sf2[k] - (nf2[k]-neq2[k])*ctau -
          cgrady*ecy[2]*(nf2[k]-nf2[nv4[k]]);
        nff3[k] = sf3[k] - (nf3[k]-neq3[k])*ctau;
        nff4[k] = sf4[k] - (nf4[k]-neq4[k])*ctau -
          cgrady*ecy[4]*(nf4[nv2[k]]-nf4[k]);
        nff5[k] = sf5[k] - (nf5[k]-neq5[k])*ctau -
          cgradx*ecx[5]*(nf5[k]-nf5[nv3[k]]) -
          cgrady*ecy[5]*(nf5[k]-nf5[nv4[k]]);
        nff6[k] = sf6[k] - (nf6[k]-neq6[k])*ctau -
          cgradx*ecx[6]*(nf6[k]-nf6[k]) -
          cgrady*ecy[6]*(nf6[k]-nf6[nv4[k]]);
        nff7[k] = sf7[k] - (nf7[k]-neq7[k])*ctau -
          cgradx*ecx[7]*(nf7[k]-nf7[k]) -
          cgrady*ecy[7]*(nf7[nv2[k]]-nf7[k]);
        nff8[k] = sf8[k] - (nf8[k]-neq8[k])*ctau -
          cgradx*ecx[8]*(nf8[k]-nf8[nv3[k]]) -
          cgrady*ecy[8]*(nf8[nv2[k]]-nf8[k]);

        /*
        nff5[k] = sf5[k] - (nf5[k]-neq5[k])*ctau -
          cgradxy*edxy*(nf5[k]-nf5[nv7[k]]);
        nff6[k] = sf6[k] - (nf6[k]-neq6[k])*ctau -
          cgradxy*edxy*(nf6[k]-nf8[k]);
        nff7[k] = sf7[k] - (nf7[k]-neq7[k])*ctau -
          cgradxy*edxy*(nf7[k]-nf5[k]);
        nff8[k] = sf8[k] - (nf8[k]-neq8[k])*ctau -
          cgradxy*edxy*(nf8[k]-nf8[nv6[k]]);
        */
        break;

        /*
```



```c
        default:
          nff0[k] = neq0[k];
          nff1[k] = neq1[k];
          nff2[k] = neq2[k];
          nff3[k] = neq3[k];
          nff4[k] = neq4[k];
          nff5[k] = neq5[k];
          nff6[k] = neq6[k];
          break;
          */
        }
    }
}


void lglb_coll(double mass, double tau, double cspeed,
               double nf0[], double nf1[], double nf2[],
               double nf3[], double nf4[], double nf5[],
               double nf6[], double nf7[], double nf8[],
               double nff0[], double nff1[], double nff2[],
               double nff3[], double nff4[], double nff5[],
               double nff6[], double nff7[], double nff8[],
               double neq0[], double neq1[], double neq2[],
               double neq3[], double neq4[], double neq5[],
               double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau = delta_t / tau;

  for(k=0; k<nnodes_all; k++)
    {
      nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
      nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau;
      nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau;
      nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau;
      nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau;
      nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau;
      nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau;
      nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau;
      nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau;
      /*
      printf("k=%d  coll\n",k);
      printf("nf0=%g  neq0=%g  nff0=%g\n",nf0[k],neq0[k],nff0[k]);
      printf("nf1=%g  neq1=%g  nff1=%g\n",nf1[k],neq1[k],nff1[k]);
      printf("nf2=%g  neq2=%g  nff2=%g\n",nf2[k],neq2[k],nff2[k]);
      printf("nf3=%g  neq3=%g  nff3=%g\n",nf3[k],neq3[k],nff3[k]);
      printf("nf4=%g  neq4=%g  nff4=%g\n",nf4[k],neq4[k],nff4[k]);
      printf("nf5=%g  neq5=%g  nff5=%g\n",nf5[k],neq5[k],nff5[k]);
      printf("nf6=%g  neq6=%g  nff6=%g\n",nf6[k],neq6[k],nff6[k]);
      printf("nf7=%g  neq7=%g  nff7=%g\n",nf7[k],neq7[k],nff7[k]);
      printf("nf8=%g  neq8=%g  nff8=%g\n",nf8[k],neq8[k],nff8[k]);
      */
    }
}

void lglb_prop(double mass, double tau, double cspeed,
                double nff0[], double nff1[], double nff2[],
                double nff3[], double nff4[], double nff5[],
                double nff6[], double nff7[], double nff8[],
                double nf0[], double nf1[], double nf2[],
                double nf3[], double nf4[], double nf5[],
                double nf6[], double nf7[], double nf8[])
{
  int k;
  double cgrad = cspeed * delta_t / delta_x;

  for(k=0; k<nnodes_all; k++)
    {
      nf0[k] = nff0[k];
```

```
    switch(boundary_mode[k])
      {
      case 10:   /*  bulk  */
        nf1[k] = nff1[k] - (nff1[k] - nff1[nv3[k]]) * cgrad;
        nf2[k] = nff2[k] - (nff2[k] - nff2[nv4[k]]) * cgrad;
        nf3[k] = nff3[k] - (nff3[k] - nff3[nv1[k]]) * cgrad;
        nf4[k] = nff4[k] - (nff4[k] - nff4[nv2[k]]) * cgrad;
        nf5[k] = nff5[k] - (nff5[k] - nff5[nv7[k]]) * cgrad;
        nf6[k] = nff6[k] - (nff6[k] - nff6[nv8[k]]) * cgrad;
        nf7[k] = nff7[k] - (nff7[k] - nff7[nv5[k]]) * cgrad;
        nf8[k] = nff8[k] - (nff8[k] - nff8[nv6[k]]) * cgrad;
        break;
      case 13:   /*  left wall  */
        nf1[k] = nff1[k] - (nff1[k] - nff3[nv1[k]]) * cgrad;
        nf2[k] = nff2[k] - (nff2[k] - nff2[nv4[k]]) * cgrad;
        nf3[k] = nff3[k] - (nff3[k] - nff3[nv1[k]]) * cgrad;
        nf4[k] = nff4[k] - (nff4[k] - nff4[nv2[k]]) * cgrad;
        nf5[k] = nff5[k] - (nff5[k] - nff7[nv5[k]]) * cgrad;
        nf6[k] = nff6[k] - (nff6[k] - nff6[nv8[k]]) * cgrad;
        nf7[k] = nff7[k] - (nff7[k] - nff7[nv5[k]]) * cgrad;
        nf8[k] = nff8[k] - (nff8[k] - nff6[nv8[k]]) * cgrad;
        break;
      case 14:   /*  right wall */
        nf1[k] = nff1[k] - (nff1[k] - nff1[nv3[k]]) * cgrad;
        nf2[k] = nff2[k] - (nff2[k] - nff2[nv4[k]]) * cgrad;
        nf3[k] = nff3[k] - (nff3[k] - nff1[nv3[k]]) * cgrad;
        nf4[k] = nff4[k] - (nff4[k] - nff4[nv2[k]]) * cgrad;
        nf5[k] = nff5[k] - (nff5[k] - nff5[nv7[k]]) * cgrad;
        nf6[k] = nff6[k] - (nff6[k] - nff8[nv6[k]]) * cgrad;
        nf7[k] = nff7[k] - (nff7[k] - nff5[nv7[k]]) * cgrad;
        nf8[k] = nff8[k] - (nff8[k] - nff8[nv6[k]]) * cgrad;
        break;
      }
    /*
    printf("k=%d  prop\n",k);
    printf("nff0=%g  nf0=%g\n",nff0[k],nf0[k]);
    printf("nff1=%g  nf1=%g\n",nff1[k],nf1[k]);
    printf("nff2=%g  nf2=%g\n",nff2[k],nf2[k]);
    printf("nff3=%g  nf3=%g\n",nff3[k],nf3[k]);
    printf("nff4=%g  nf4=%g\n",nff4[k],nf4[k]);
    printf("nff5=%g  nf5=%g\n",nff5[k],nf5[k]);
    printf("nff6=%g  nf6=%g\n",nff6[k],nf6[k]);
    printf("nff7=%g  nf7=%g\n",nff7[k],nf7[k]);
    printf("nff8=%g  nf8=%g\n",nff8[k],nf8[k]);
    */
  }
}

void convection_diffusion(double mass, double tau, double cspeed,
                          double nf0[], double nf1[], double nf2[],
                          double nf3[], double nf4[], double nf5[],
                          double nf6[], double nf7[], double nf8[],
                          double nff0[], double nff1[], double nff2[],
                          double nff3[], double nff4[], double nff5[],
                          double nff6[], double nff7[], double nff8[],
                          double neq0[], double neq1[], double neq2[],
                          double neq3[], double neq4[], double neq5[],
                          double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxs2, cgradys2, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double delta_x2, delta_y2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
```



```
  cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_x;
  cgradys2 = cspeed * sqrt((double) 2) * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
              (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  delta_x2 = delta_x * delta_x;
  delta_y2 = delta_y * delta_y;

  for(k=0; k<nnodes_all; k++)
   {
    switch(boundary_mode[k])
      {
      case 0:   /*  bulk  */
        nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
          dcoef * (
                  (nf0[nv3[k]] - 2.000*nf0[k] + nf0[nv1[k]]) / delta_x2 +
                  (nf0[nv2[k]] - 2.000*nf0[k] + nf0[nv4[k]]) / delta_y2 );
        nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau -
          cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]) +
          dcoef * (
                  (nf1[nv3[k]] - 2.000*nf1[k] + nf1[nv1[k]]) / delta_x2 +
                  (nf1[nv2[k]] - 2.000*nf1[k] + nf1[nv4[k]]) / delta_y2 );
        nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau -
          cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]) +
          dcoef * (
                  (nf2[nv3[k]] - 2.000*nf2[k] + nf2[nv1[k]]) / delta_x2 +
                  (nf2[nv2[k]] - 2.000*nf2[k] + nf2[nv4[k]]) / delta_y2 );
        nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau -
          cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]) +
          dcoef * (
                  (nf3[nv3[k]] - 2.000*nf3[k] + nf3[nv1[k]]) / delta_x2 +
                  (nf3[nv2[k]] - 2.000*nf3[k] + nf3[nv4[k]]) / delta_y2 );
        nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau -
          cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]) +
          dcoef * (
                  (nf4[nv3[k]] - 2.000*nf4[k] + nf4[nv1[k]]) / delta_x2 +
                  (nf4[nv2[k]] - 2.000*nf4[k] + nf4[nv4[k]]) / delta_y2 );
        nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau -
          cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]]) -
          cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]) +
          dcoef * (
                  (nf5[nv3[k]] - 2.000*nf5[k] + nf5[nv1[k]]) / delta_x2 +
                  (nf5[nv2[k]] - 2.000*nf5[k] + nf5[nv4[k]]) / delta_y2 );
        nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau -
          cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k]) -
          cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]) +
          dcoef * (
                  (nf6[nv3[k]] - 2.000*nf6[k] + nf6[nv1[k]]) / delta_x2 +
                  (nf6[nv2[k]] - 2.000*nf6[k] + nf6[nv4[k]]) / delta_y2 );
        nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau -
          cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k]) -
          cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]) +
          dcoef * (
                  (nf7[nv3[k]] - 2.000*nf7[k] + nf7[nv1[k]]) / delta_x2 +
                  (nf7[nv2[k]] - 2.000*nf7[k] + nf7[nv4[k]]) / delta_y2 );
        nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau -
          cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]]) -
          cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) +
          dcoef * (
                  (nf8[nv3[k]] - 2.000*nf8[k] + nf8[nv1[k]]) / delta_x2 +
                  (nf8[nv2[k]] - 2.000*nf8[k] + nf8[nv4[k]]) / delta_y2 );
        /*
        printf("k==%d     %g  %g  %g  %g  %g  %g  %g  %g  %g  %g  %g\n",k,
               neq0[k],neq1[k],neq2[k],neq3[k],neq4[k],neq5[k],
               neq6[k],neq7[k],neq8[k],uxloc[k],uyloc[k]);
        printf("k==%d   nf %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
```

```
          k,nf0[k],nf1[k],nf2[k],nf3[k],nf4[k],nf5[k],
          nf6[k],nf7[k],nf8[k]);
   printf("k==%d  nff %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
          k,nff0[k],nff1[k],nff2[k],nff3[k],nff4[k],nff5[k],
          nff6[k],nff7[k],nff8[k]);
   */
   break;

 case 3:   /*  left  wall  */

   nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
     dcoef * (
              ( 2.000*nf0[k] - 5.000*nf0[nv1[k]] +
                4.000*nf0[nv1[nv1[k]]] -
                nf0[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf0[nv2[k]] - 2.000*nf0[k] + nf0[nv4[k]]) / delta_y2 );
   nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
     - cgradx * ecx[1] * (nf1[k] - nf1[k]) +
     dcoef * (
              ( 2.000*nf1[k] - 5.000*nf1[nv1[k]] +
                4.000*nf1[nv1[nv1[k]]] -
                nf1[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf1[nv2[k]] - 2.000*nf1[k] + nf1[nv4[k]]) / delta_y2 );
   nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
     - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]) +
     dcoef * (
              ( 2.000*nf2[k] - 5.000*nf2[nv1[k]] +
                4.000*nf2[nv1[nv1[k]]] -
                nf2[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf2[nv2[k]] - 2.000*nf2[k] + nf2[nv4[k]]) / delta_y2 );
   nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
     - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]) +
     dcoef * (
              ( 2.000*nf3[k] - 5.000*nf3[nv1[k]] +
                4.000*nf3[nv1[nv1[k]]] -
                nf3[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf3[nv2[k]] - 2.000*nf3[k] + nf3[nv4[k]]) / delta_y2 );
   nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
     - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]) +
     dcoef * (
              ( 2.000*nf4[k] - 5.000*nf4[nv1[k]] +
                4.000*nf4[nv1[nv1[k]]] -
                nf4[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf4[nv2[k]] - 2.000*nf4[k] + nf4[nv4[k]]) / delta_y2 );
   nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
     - cgradx * ecx[5] * (nf5[k] - nf5[k])
     - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]) +
     dcoef * (
              ( 2.000*nf5[k] - 5.000*nf5[nv1[k]] +
                4.000*nf5[nv1[nv1[k]]] -
                nf5[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf5[nv2[k]] - 2.000*nf5[k] + nf5[nv5[k]]) / delta_y2 );
   nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
     - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
     - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]) +
     dcoef * (
              ( 2.000*nf6[k] - 5.000*nf6[nv1[k]] +
                4.000*nf6[nv1[nv1[k]]] -
                nf6[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf6[nv2[k]] - 2.000*nf6[k] + nf6[nv4[k]]) / delta_y2 );
   nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
     - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
     - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]) +
     dcoef * (
              ( 2.000*nf7[k] - 5.000*nf7[nv1[k]] +
                4.000*nf7[nv1[nv1[k]]] -
                nf7[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf7[nv2[k]] - 2.000*nf7[k] + nf7[nv4[k]]) / delta_y2 );
   nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
```



```
     - cgradx * ecx[8] * (nf8[k] - nf8[k])
     - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) +
     dcoef * (
              ( 2.000*nf8[k] - 5.000*nf8[nv1[k]] +
                4.000*nf8[nv1[nv1[k]]] -
                nf8[nv1[nv1[nv1[k]]]] ) / delta_x2 +
              (nf8[nv2[k]] - 2.000*nf8[k] + nf8[nv4[k]]) / delta_y2 );
   break;

 case 4:  /*  right  wall */

   nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +
     dcoef * (
              ( 2.000*nf0[k] - 5.000*nf0[nv3[k]] +
                4.000*nf0[nv3[nv3[k]]] -
                nf0[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf0[nv2[k]] - 2.000*nf0[k] + nf0[nv4[k]]) / delta_y2 );
   nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
     - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]) +
     dcoef * (
              ( 2.000*nf1[k] - 5.000*nf1[nv3[k]] +
                4.000*nf1[nv3[nv3[k]]] -
                nf1[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf1[nv2[k]] - 2.000*nf1[k] + nf1[nv4[k]]) / delta_y2 );
   nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
     - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]) +
     dcoef * (
              ( 2.000*nf2[k] - 5.000*nf2[nv3[k]] +
                4.000*nf2[nv3[nv3[k]]] -
                nf2[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf2[nv2[k]] - 2.000*nf2[k] + nf2[nv4[k]]) / delta_y2 );
   nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
     - cgradx * ecx[3] * (nf3[k] - nf3[k]) +
     dcoef * (
              ( 2.000*nf3[k] - 5.000*nf3[nv3[k]] +
                4.000*nf3[nv3[nv3[k]]] -
                nf3[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf3[nv2[k]] - 2.000*nf3[k] + nf3[nv4[k]]) / delta_y2 );
   nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
     - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]) +
     dcoef * (
              ( 2.000*nf4[k] - 5.000*nf4[nv3[k]] +
                4.000*nf4[nv3[nv3[k]]] -
                nf4[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf4[nv2[k]] - 2.000*nf4[k] + nf4[nv4[k]]) / delta_y2 );
   nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
     - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
     - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]) +
     dcoef * (
              ( 2.000*nf5[k] - 5.000*nf5[nv3[k]] +
                4.000*nf5[nv3[nv3[k]]] -
                nf5[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf5[nv2[k]] - 2.000*nf5[k] + nf5[nv4[k]]) / delta_y2 );
   nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
     - cgradx * ecx[6] * (nf6[k] - nf6[k])
     - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]) +
     dcoef * (
              ( 2.000*nf6[k] - 5.000*nf6[nv3[k]] +
                4.000*nf6[nv3[nv3[k]]] -
                nf6[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf6[nv2[k]] - 2.000*nf6[k] + nf6[nv4[k]]) / delta_y2 );
   nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
     - cgradx * ecx[7] * (nf7[k] - nf7[k])
     - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]) +
     dcoef * (
              ( 2.000*nf7[k] - 5.000*nf7[nv3[k]] +
                4.000*nf7[nv3[nv3[k]]] -
                nf7[nv3[nv3[nv3[k]]]] ) / delta_x2 +
              (nf7[nv2[k]] - 2.000*nf7[k] + nf7[nv4[k]]) / delta_y2 );
```

```
            nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
              - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
              - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) +
              dcoef * (
                      ( 2.000*nf8[k] - 5.000*nf8[nv3[k]] +
                        4.000*nf8[nv3[nv3[k]]] -
                        nf8[nv3[nv3[nv3[k]]]] ) / delta_x2 +
                      (nf8[nv2[k]] - 2.000*nf8[k] + nf8[nv4[k]]) / delta_y2 );
            break;
          }
      }
}

void first_upwind(double mass, double tau, double cspeed,
                  double nf0[], double nf1[], double nf2[],
                  double nf3[], double nf4[], double nf5[],
                  double nf6[], double nf7[], double nf8[],
                  double nff0[], double nff1[], double nff2[],
                  double nff3[], double nff4[], double nff5[],
                  double nff6[], double nff7[], double nff8[],
                  double neq0[], double neq1[], double neq2[],
                  double neq3[], double neq4[], double neq5[],
                  double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxs2, cgradys2, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_x;
  cgradys2 = cspeed * sqrt((double) 2) * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                  (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));


  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */
          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
            - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
            - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
            - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
            - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
          /*
```



```
          printf("k==%d      %g  %g  %g  %g  %g  %g  %g  %g  %g  %g  %g\n",k,
                 neq0[k],neq1[k],neq2[k],neq3[k],neq4[k],neq5[k],
                 neq6[k],neq7[k],neq8[k],uxloc[k],uyloc[k]);
          printf("k==%d   nf %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
                 k,nf0[k],nf1[k],nf2[k],nf3[k],nf4[k],nf5[k],
                 nf6[k],nf7[k],nf8[k]);
          printf("k==%d  nff %g  %g  %g  %g  %g  %g  %g  %g  %g\n",
                 k,nff0[k],nff1[k],nff2[k],nff3[k],nff4[k],nff5[k],
                 nff6[k],nff7[k],nff8[k]);
          */
          break;

        case 3:   /*  left  wall  */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx * ecx[1] * (nf1[k] - nf1[k]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx * ecx[3] * (nf3[nv1[k]] - nf3[k]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx * ecx[5] * (nf5[k] - nf5[k])
            - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k])
            - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx * ecx[7] * (nf7[nv1[k]] - nf7[k])
            - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx * ecx[8] * (nf8[k] - nf8[k])
            - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
          break;

        case 4:   /*  right  wall */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx * ecx[1] * (nf1[k] - nf1[nv3[k]]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx * ecx[3] * (nf3[k] - nf3[k]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]])
            - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx * ecx[6] * (nf6[k] - nf6[k])
            - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx * ecx[7] * (nf7[k] - nf7[k])
            - cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]])
            - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);
          break;
        }
    }
}

void second_upwind(double mass, double tau, double cspeed,
                   double nf0[], double nf1[], double nf2[],
                   double nf3[], double nf4[], double nf5[],
                   double nf6[], double nf7[], double nf8[],
```

```c
                    double nff0[], double nff1[], double nff2[],
                    double nff3[], double nff4[], double nff5[],
                    double nff6[], double nff7[], double nff8[],
                    double neq0[], double neq1[], double neq2[],
                    double neq3[], double neq4[], double neq5[],
                    double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                    (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */
          break;
        }
    }
}

void first_centered(double mass, double tau, double cspeed,
                    double nf0[], double nf1[], double nf2[],
                    double nf3[], double nf4[], double nf5[],
                    double nf6[], double nf7[], double nf8[],
                    double nff0[], double nff1[], double nff2[],
                    double nff3[], double nff4[], double nff5[],
                    double nff6[], double nff7[], double nff8[],
                    double neq0[], double neq1[], double neq2[],
                    double neq3[], double neq4[], double neq5[],
                    double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                    (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */
          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[nv3[k]]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
```



```c
            - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[nv3[k]]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx2 * ecx[5] * (nf5[nv1[k]] - nf5[nv3[k]])
            - cgradxy * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx2 * ecx[6] * (nf6[nv1[k]] - nf6[nv3[k]])
            - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx2 * ecx[7] * (nf7[nv1[k]] - nf7[nv3[k]])
            - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[nv3[k]])
            - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
          break;

        case 3:   /*  left  wall  */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[k]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[k]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx2 * ecx[5] * (nf5[nv1[k]] - nf5[k])
            - cgradxy * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx2 * ecx[6] * (nf6[nv1[k]] - nf6[k])
            - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx2 * ecx[7] * (nf7[nv1[k]] - nf7[k])
            - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[k])
            - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
          break;

        case 4:  /*  right  wall */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - cgradx2 * ecx[1] * (nf1[k] - nf1[nv3[k]]);
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - cgradx2 * ecx[3] * (nf3[k] - nf3[nv3[k]]);
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - cgradx2 * ecx[5] * (nf5[k] - nf5[nv3[k]])
            - cgradxy * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - cgradx2 * ecx[6] * (nf6[k] - nf6[nv3[k]])
            - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - cgradx2 * ecx[7] * (nf7[k] - nf7[nv3[k]])
            - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - cgradx2 * ecx[8] * (nf8[k] - nf8[nv3[k]])
            - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
          break;
        }
    }
}
```

```
void lax_friedrichs(double mass, double tau, double cspeed,
                    double nf0[], double nf1[], double nf2[],
                    double nf3[], double nf4[], double nf5[],
                    double nf6[], double nf7[], double nf8[],
                    double nff0[], double nff1[], double nff2[],
                    double nff3[], double nff4[], double nff5[],
                    double nff6[], double nff7[], double nff8[],
                    double neq0[], double neq1[], double neq2[],
                    double neq3[], double neq4[], double neq5[],
                    double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */

          nff0[k] = (nf0[nv1[k]]+nf0[nv3[k]]+nf0[nv2[k]]+nf0[nv4[k]]) / 4.000
            - (nf0[k]-neq0[k])*ctau;
          nff1[k] = (nf1[nv1[k]]+nf1[nv3[k]]+nf1[nv2[k]]+nf1[nv4[k]]) / 4.000
            - (nf1[k]-neq1[k])*ctau
            - cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[nv3[k]]);
          nff2[k] = (nf2[nv1[k]]+nf2[nv3[k]]+nf2[nv2[k]]+nf2[nv4[k]]) / 4.000
            - (nf2[k]-neq2[k])*ctau
            - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
          nff3[k] = (nf3[nv1[k]]+nf3[nv3[k]]+nf3[nv2[k]]+nf3[nv4[k]]) / 4.000
            - (nf3[k]-neq3[k])*ctau
            - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[nv3[k]]);
          nff4[k] = (nf4[nv1[k]]+nf4[nv3[k]]+nf4[nv2[k]]+nf4[nv4[k]]) / 4.000
            - (nf4[k]-neq4[k])*ctau
            - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
          nff5[k] = (nf5[nv1[k]]+nf5[nv2[k]]+nf5[nv3[k]]+nf5[nv4[k]]) / 4.000
            - (nf5[k]-neq5[k])*ctau
            - cgradx2 * ecx[5] * (nf5[nv1[k]] - nf5[nv3[k]])
            - cgrady2 * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
          nff6[k] = (nf6[nv1[k]]+nf6[nv2[k]]+nf6[nv3[k]]+nf6[nv4[k]]) / 4.000
            - (nf6[k]-neq6[k])*ctau
            - cgradx2 * ecx[6] * (nf6[nv1[k]] - nf6[nv3[k]])
            - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
          nff7[k] = (nf7[nv1[k]]+nf7[nv2[k]]+nf7[nv3[k]]+nf7[nv4[k]]) / 4.000
            - (nf7[k]-neq7[k])*ctau
            - cgradx2 * ecx[7] * (nf7[nv1[k]] - nf7[nv3[k]])
            - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
          nff8[k] = (nf8[nv1[k]]+nf8[nv2[k]]+nf8[nv3[k]]+nf8[nv4[k]]) / 4.000
            - (nf8[k]-neq8[k])*ctau
            - cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[nv3[k]])
            - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
          break;

        case 3:   /*  left  wall  */

          nff0[k] = (nf0[nv1[k]]+nf0[k]+nf0[nv2[k]]+nf0[nv4[k]]) / 4.000
            - (nf0[k]-neq0[k])*ctau;
```



```
          nff1[k] = (nf1[nv1[k]]+nf1[k]+nf1[nv2[k]]+nf1[nv4[k]]) / 4.000
            - (nf1[k]-neq1[k])*ctau
            - cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[k]);
          nff2[k] = (nf2[nv1[k]]+nf2[k]+nf2[nv2[k]]+nf2[nv4[k]]) / 4.000
            - (nf2[k]-neq2[k])*ctau
            - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
          nff3[k] = (nf3[nv1[k]]+nf3[k]+nf3[nv2[k]]+nf3[nv4[k]]) / 4.000
            - (nf3[k]-neq3[k])*ctau
            - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[k]);
          nff4[k] = (nf4[nv1[k]]+nf4[k]+nf4[nv2[k]]+nf4[nv4[k]]) / 4.000
            - (nf4[k]-neq4[k])*ctau
            - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
          nff5[k] = (nf5[nv1[k]]+nf5[nv2[k]]+nf5[k]+nf5[nv4[k]]) / 4.000
            - (nf5[k]-neq5[k])*ctau
            - cgradx2 * ecx[5] * (nf5[nv1[k]] - nf5[k])
            - cgrady2 * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
          nff6[k] = (nf6[nv1[k]]+nf6[nv2[k]]+nf6[k]+nf6[nv4[k]]) / 4.000
            - (nf6[k]-neq6[k])*ctau
            - cgradx2 * ecx[6] * (nf6[nv1[k]] - nf6[k])
            - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
          nff7[k] = (nf7[nv1[k]]+nf7[nv2[k]]+nf7[k]+nf7[nv4[k]]) / 4.000
            - (nf7[k]-neq7[k])*ctau
            - cgradx2 * ecx[7] * (nf7[nv1[k]] - nf7[k])
            - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
          nff8[k] = (nf8[nv1[k]]+nf8[nv2[k]]+nf8[k]+nf8[nv4[k]]) / 4.000
            - (nf8[k]-neq8[k])*ctau
            - cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[k])
            - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
          break;

        case 4:   /*  right  wall  */

          nff0[k] = (nf0[k]+nf0[nv3[k]]+nf0[nv2[k]]+nf0[nv4[k]]) / 4.000
            - (nf0[k]-neq0[k])*ctau;
          nff1[k] = (nf1[k]+nf1[nv3[k]]+nf1[nv2[k]]+nf1[nv4[k]]) / 4.000
            - (nf1[k]-neq1[k])*ctau
            - cgradx2 * ecx[1] * (nf1[k] - nf1[nv3[k]]);
          nff2[k] = (nf2[k]+nf2[nv3[k]]+nf2[nv2[k]]+nf2[nv4[k]]) / 4.000
            - (nf2[k]-neq2[k])*ctau
            - cgrady2 * ecy[2] * (nf2[nv2[k]] - nf2[nv4[k]]);
          nff3[k] = (nf3[k]+nf3[nv3[k]]+nf3[nv2[k]]+nf3[nv4[k]]) / 4.000
            - (nf3[k]-neq3[k])*ctau
            - cgradx2 * ecx[3] * (nf3[k] - nf3[nv3[k]]);
          nff4[k] = (nf4[k]+nf4[nv3[k]]+nf4[nv2[k]]+nf4[nv4[k]]) / 4.000
            - (nf4[k]-neq4[k])*ctau
            - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]);
          nff5[k] = (nf5[k]+nf5[nv2[k]]+nf5[nv3[k]]+nf5[nv4[k]]) / 4.000
            - (nf5[k]-neq5[k])*ctau
            - cgradx2 * ecx[5] * (nf5[k] - nf5[nv3[k]])
            - cgrady2 * ecy[5] * (nf5[nv2[k]] - nf5[nv4[k]]);
          nff6[k] = (nf6[k]+nf6[nv2[k]]+nf6[nv3[k]]+nf6[nv4[k]]) / 4.000
            - (nf6[k]-neq6[k])*ctau
            - cgradx2 * ecx[6] * (nf6[k] - nf6[nv3[k]])
            - cgrady2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]);
          nff7[k] = (nf7[k]+nf7[nv2[k]]+nf7[nv3[k]]+nf7[nv4[k]]) / 4.000
            - (nf7[k]-neq7[k])*ctau
            - cgradx2 * ecx[7] * (nf7[k] - nf7[nv3[k]])
            - cgrady2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);
          nff8[k] = (nf8[k]+nf8[nv2[k]]+nf8[nv3[k]]+nf8[nv4[k]]) / 4.000
            - (nf8[k]-neq8[k])*ctau
            - cgradx2 * ecx[8] * (nf8[k] - nf8[nv3[k]])
            - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]);
          break;

        }
    }
}

void lax_wendroff(double mass, double tau, double cspeed,
```

```c
                  double nf0[], double nf1[], double nf2[],
                  double nf3[], double nf4[], double nf5[],
                  double nf6[], double nf7[], double nf8[],
                  double nff0[], double nff1[], double nff2[],
                  double nff3[], double nff4[], double nff5[],
                  double nff6[], double nff7[], double nff8[],
                  double neq0[], double neq1[], double neq2[],
                  double neq3[], double neq4[], double neq5[],
                  double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxy, xdummy, ydummy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                   (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          xdummy = cgradx2 * ecx[1];
          ydummy = cgrady2 * ecy[1];
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - xdummy * (nf1[nv1[k]] - nf1[nv3[k]])
            + xdummy * cgradx * ecx[1]
            * (nf1[nv1[k]] - 2.000 * nf1[k] + nf1[nv3[k]]);
          xdummy = cgradx2 * ecx[2];
          ydummy = cgrady2 * ecy[2];
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - ydummy * (nf2[nv2[k]] - nf2[nv4[k]])
            + ydummy * cgrady * ecy[2]
            * (nf2[nv2[k]] - 2.000 * nf2[k] + nf2[nv4[k]]);
          xdummy = cgradx2 * ecx[3];
          ydummy = cgrady2 * ecy[3];
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - xdummy * (nf3[nv1[k]] - nf3[nv3[k]])
            + xdummy * cgradx * ecx[3]
            * (nf3[nv1[k]] - 2.000 * nf3[k] + nf3[nv3[k]]);
          xdummy = cgradx2 * ecx[4];
          ydummy = cgrady2 * ecy[4];
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - ydummy * (nf4[nv2[k]] - nf4[nv4[k]])
            + ydummy * cgrady * ecy[4]
            * (nf4[nv2[k]] - 2.000 * nf4[k] + nf4[nv4[k]]);
          xdummy = cgradx2 * ecx[5];
          ydummy = cgrady2 * ecy[5];
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - xdummy * (nf5[nv1[k]] - nf5[nv3[k]])
            + xdummy * cgradx * ecx[5]
            * (nf5[nv1[k]] - 2.000 * nf5[k] + nf5[nv3[k]])
            - ydummy * (nf5[nv2[k]] - nf5[nv4[k]])
            + ydummy * cgrady * ecy[5]
            * (nf5[nv2[k]] - 2.000 * nf5[k] + nf5[nv4[k]]);
          xdummy = cgradx2 * ecx[6];
          ydummy = cgrady2 * ecy[6];
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - xdummy * (nf6[nv1[k]] - nf6[nv3[k]])
```



```c
            + xdummy * cgradx * ecx[6]
            * (nf6[nv1[k]] - 2.000 * nf6[k] + nf6[nv3[k]])
            - ydummy * (nf6[nv2[k]] - nf6[nv4[k]])
            + ydummy * cgrady * ecy[6]
            * (nf6[nv2[k]] - 2.000 * nf6[k] + nf6[nv4[k]]);
          xdummy = cgradx2 * ecx[7];
          ydummy = cgrady2 * ecy[7];
          nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
            - xdummy * (nf7[nv1[k]] - nf7[nv3[k]])
            + xdummy * cgradx * ecx[7]
            * (nf7[nv1[k]] - 2.000 * nf7[k] + nf7[nv3[k]])
            - ydummy * (nf7[nv2[k]] - nf7[nv4[k]])
            + ydummy * cgrady * ecy[7]
            * (nf7[nv2[k]] - 2.000 * nf7[k] + nf7[nv4[k]]);
          xdummy = cgradx2 * ecx[8];
          ydummy = cgrady2 * ecy[8];
          nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
            - xdummy * (nf8[nv1[k]] - nf8[nv3[k]])
            + xdummy * cgradx * ecx[8]
            * (nf8[nv1[k]] - 2.000 * nf8[k] + nf8[nv3[k]])
            - ydummy * (nf8[nv2[k]] - nf8[nv4[k]])
            + ydummy * cgrady * ecy[8]
            * (nf8[nv2[k]] - 2.000 * nf8[k] + nf8[nv4[k]]);
          break;

        case 3:   /*  left  wall  */

          nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
          xdummy = cgradx2 * ecx[1];
          ydummy = cgradx2 * ecy[1];
          nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
            - xdummy * (nf1[nv1[k]] - nf1[k])
            + xdummy * cgradx * ecx[1]
            * (nf1[nv1[k]] - 2.000 * nf1[k] + nf1[k]);
          xdummy = cgradx2 * ecx[2];
          ydummy = cgrady2 * ecy[2];
          nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
            - ydummy * (nf2[nv2[k]] - nf2[nv4[k]])
            + ydummy * cgrady * ecy[2]
            * (nf2[nv2[k]] - 2.000 * nf2[k] + nf2[nv4[k]]);
          xdummy = cgradx2 * ecx[3];
          ydummy = cgrady2 * ecy[3];
          nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
            - xdummy * (nf3[nv1[k]] - nf3[k])
            + xdummy * cgradx * ecx[3]
            * (nf3[nv1[k]] - 2.000 * nf3[k] + nf3[k]);
          xdummy = cgradx2 * ecx[4];
          ydummy = cgrady2 * ecy[4];
          nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
            - ydummy * (nf4[nv2[k]] - nf4[nv4[k]])
            + ydummy * cgrady * ecy[4]
            * (nf4[nv2[k]] - 2.000 * nf4[k] + nf4[nv4[k]]);
          xdummy = cgradx2 * ecx[5];
          ydummy = cgrady2 * ecy[5];
          nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
            - xdummy * (nf5[nv1[k]] - nf5[k])
            + xdummy * cgradx * ecx[5]
            * (nf5[nv1[k]] - 2.000 * nf5[k] + nf5[k])
            - ydummy * (nf5[nv2[k]] - nf5[nv4[k]])
            + ydummy * cgrady * ecy[5]
            * (nf5[nv2[k]] - 2.000 * nf5[k] + nf5[nv4[k]]);
          xdummy = cgradx2 * ecx[6];
          ydummy = cgrady2 * ecy[6];
          nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
            - xdummy * (nf6[nv1[k]] - nf6[k])
            + xdummy * cgradx * ecx[6]
            * (nf6[nv1[k]] - 2.000 * nf6[k] + nf6[k])
            - ydummy * (nf6[nv2[k]] - nf6[nv4[k]])
            + ydummy * cgrady * ecy[6]
```

```
        * (nf6[nv2[k]] - 2.000 * nf6[k] + nf6[nv4[k]]);
      xdummy = cgradx2 * ecx[7];
      ydummy = cgrady2 * ecy[7];
      nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
        - xdummy * (nf7[nv1[k]] - nf7[k])
        + xdummy * cgradx * ecx[7]
        * (nf7[nv1[k]] - 2.000 * nf7[k] + nf7[k])
        - ydummy * (nf7[nv2[k]] - nf7[nv4[k]])
        + ydummy * cgrady * ecy[7]
        * (nf7[nv2[k]] - 2.000 * nf7[k] + nf7[nv4[k]]);
      xdummy = cgradx2 * ecx[8];
      ydummy = cgrady2 * ecy[8];
      nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
        - xdummy * (nf8[nv1[k]] - nf8[k])
        + xdummy * cgradx * ecx[8]
        * (nf8[nv1[k]] - 2.000 * nf8[k] + nf8[k])
        - ydummy * (nf8[nv2[k]] - nf8[nv4[k]])
        + ydummy * cgrady * ecy[8]
        * (nf8[nv2[k]] - 2.000 * nf8[k] + nf8[nv4[k]]);
      break;

    case 4:   /*  right  wall  */

      nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;
      xdummy = cgradx2 * ecx[1];
      ydummy = cgrady2 * ecy[1];
      nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau
        - xdummy * (nf1[k] - nf1[nv3[k]])
        + xdummy * cgradx * ecx[1]
        * (nf1[k] - 2.000 * nf1[k] + nf1[nv3[k]]);
      xdummy = cgradx2 * ecx[2];
      ydummy = cgrady2 * ecy[2];
      nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau
        - ydummy * (nf2[nv2[k]] - nf2[nv4[k]])
        + ydummy * cgrady * ecy[2]
        * (nf2[nv2[k]] - 2.000 * nf2[k] + nf2[nv4[k]]);
      xdummy = cgradx2 * ecx[3];
      ydummy = cgrady2 * ecy[3];
      nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau
        - xdummy * (nf3[k] - nf3[nv3[k]])
        + xdummy * cgradx * ecx[3]
        * (nf3[k] - 2.000 * nf3[k] + nf3[nv3[k]]);
      xdummy = cgradx2 * ecx[4];
      ydummy = cgrady2 * ecy[4];
      nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau
        - ydummy * (nf4[nv2[k]] - nf4[nv4[k]])
        + ydummy * cgrady * ecy[4]
        * (nf4[nv2[k]] - 2.000 * nf4[k] + nf4[nv4[k]]);
      xdummy = cgradx2 * ecx[5];
      ydummy = cgrady2 * ecy[5];
      nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau
        - xdummy * (nf5[k] - nf5[nv3[k]])
        + xdummy * cgradx * ecx[5]
        * (nf5[k] - 2.000 * nf5[k] + nf5[nv3[k]])
        - ydummy * (nf5[nv2[k]] - nf5[nv4[k]])
        + ydummy * cgrady * ecy[5]
        * (nf5[nv2[k]] - 2.000 * nf5[k] + nf5[nv4[k]]);
      xdummy = cgradx2 * ecx[6];
      ydummy = cgrady2 * ecy[6];
      nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau
        - xdummy * (nf6[k] - nf6[nv3[k]])
        + xdummy * cgradx * ecx[6]
        * (nf6[k] - 2.000 * nf6[k] + nf6[nv3[k]])
        - ydummy * (nf6[nv2[k]] - nf6[nv4[k]])
        + ydummy * cgrady * ecy[6]
        * (nf6[nv2[k]] - 2.000 * nf6[k] + nf6[nv4[k]]);
      xdummy = cgradx2 * ecx[7];
      ydummy = cgrady2 * ecy[7];
      nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau
```



```
          - xdummy * (nf7[k] - nf7[nv3[k]])
          + xdummy * cgradx * ecx[7]
          * (nf7[k] - 2.000 * nf7[k] + nf7[nv3[k]])
          - ydummy * (nf7[nv2[k]] - nf7[nv4[k]])
          + ydummy * cgrady * ecy[7]
          * (nf7[nv2[k]] - 2.000 * nf7[k] + nf7[nv4[k]]);
        xdummy = cgradx2 * ecx[8];
        ydummy = cgrady2 * ecy[8];
        nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau
          - xdummy * (nf8[k] - nf8[nv3[k]])
          + xdummy * cgradx * ecx[8]
          * (nf8[k] - 2.000 * nf8[k] + nf8[nv3[k]])
          - ydummy * (nf8[nv2[k]] - nf8[nv4[k]])
          + ydummy * cgrady * ecy[8]
          * (nf8[nv2[k]] - 2.000 * nf8[k] + nf8[nv4[k]]);
        break;
      }
    }
}

void leapfrog(double mass, double tau, double cspeed,
              double nf0[], double nf1[], double nf2[],
              double nf3[], double nf4[], double nf5[],
              double nf6[], double nf7[], double nf8[],
              double nff0[], double nff1[], double nff2[],
              double nff3[], double nff4[], double nff5[],
              double nff6[], double nff7[], double nff8[],
              double neq0[], double neq1[], double neq2[],
              double neq3[], double neq4[], double neq5[],
              double neq6[], double neq7[], double neq8[])
{
  int k;
  double ctau, cgrad, cgradx, cgrady, cgradxy;
  double cgradx2, cgrady2, cgradxy2;
  double dummy_force;
  double prodscal;

  ctau = delta_t / tau;
  cgradx = cspeed * delta_t / delta_x;
  cgrady = cspeed * delta_t / delta_y;
  cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y);
  cgradx2 = cspeed * delta_t / (2.000*delta_x);
  cgrady2 = cspeed * delta_t / (2.000*delta_y);
  cgradxy2 = cspeed * delta_t /
                  (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));

  for(k=0; k<nnodes_all; k++)
    {
      switch(boundary_mode[k])
        {
        case 0:   /*  bulk  */
          break;
        }
    }
}

void dif9_automaton(void)
{
  int autiter, key_point;
  autiter = 0;

  while( autiter < niter_cycle )
    {
      test_distribution_functions(f10,f11,f12,f13,f14,f15,f16,f17,f18);
      test_distribution_functions(f20,f21,f22,f23,f24,f25,f26,f26,f28);
      compute_local_speeds(f10,f11,f12,f13,f14,f15,f16,f17,f18,
                           f20,f21,f22,f23,f24,f25,f26,f27,f28);
      compute_equilibrium_distributions();
```

```
  switch(key_scheme)
    {
    case 0:
    case 10:
      first_upwind(mass1, tau1, cspeed1,
                   f10, f11, f12, f13, f14, f15, f16, f17, f18,
                   ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                   neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                   neq17, neq18);
      first_upwind(mass2, tau2, cspeed2,
                   f20, f21, f22, f23, f24, f25, f26, f27, f28,
                   ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                   neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                   neq27, neq28);
      break;
    case 1:
      second_upwind(mass1, tau1, cspeed1,
                    f10, f11, f12, f13, f14, f15, f16, f17, f18,
                    ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                    neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                    neq17, neq18);
      second_upwind(mass2, tau2, cspeed2,
                    f20, f21, f22, f23, f24, f25, f26, f27, f28,
                    ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                    neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                    neq27, neq28);
      break;
    case 2:
      first_centered(mass1, tau1, cspeed1,
                     f10, f11, f12, f13, f14, f15, f16, f17, f18,
                     ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                     neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                     neq17, neq18);
      first_centered(mass2, tau2, cspeed2,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                     neq27, neq28);
      break;
    case 3:
      lax_friedrichs(mass1, tau1, cspeed1,
                     f10, f11, f12, f13, f14, f15, f16, f17, f18,
                     ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                     neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                     neq17, neq18);
      lax_friedrichs(mass2, tau2, cspeed2,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                     neq27, neq28);
      break;
    case 4:
      lax_wendroff(mass1, tau1, cspeed1,
                   f10, f11, f12, f13, f14, f15, f16, f17, f18,
                   ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                   neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                   neq17, neq18);
      lax_wendroff(mass2, tau2, cspeed2,
                   f20, f21, f22, f23, f24, f25, f26, f27, f28,
                   ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                   neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                 neq27, neq28);
      break;
    case 5:
      leapfrog(mass1, tau1, cspeed1,
               f10, f11, f12, f13, f14, f15, f16, f17, f18,
               ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
               neq10, neq11, neq12, neq13, neq14, neq15, neq16,
               neq17, neq18);
```



```
      leapfrog(mass2, tau2, cspeed2,
               f20, f21, f22, f23, f24, f25, f26, f27, f28,
               ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
               neq20, neq21, neq22, neq23, neq24, neq25, neq26,
               neq27, neq28);
      break;
    case 6:
      store_distribution_functions();
      key_point = 0;
      runge_kutta(key_point, mass1, tau1, cspeed1,
                  f10, f11, f12, f13, f14, f15, f16, f17, f18,
                  ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                  sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
                  neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                  neq17, neq18);
      runge_kutta(key_point, mass2, tau2, cspeed2,
                  f20, f21, f22, f23, f24, f25, f26, f27, f28,
                  ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                  sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
                  neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                  neq27, neq28);
      break;
    case 7:
      lglb_coll(mass1, tau1, cspeed1,
                f10, f11, f12, f13, f14, f15, f16, f17, f18,
                ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                neq17, neq18);
      lglb_coll(mass2, tau2, cspeed2,
                f20, f21, f22, f23, f24, f25, f26, f27, f28,
                ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                neq27, neq28);
      lglb_prop(mass1, tau1, cspeed1,
                ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                f10, f11, f12, f13, f14, f15, f16, f17, f18);
      lglb_prop(mass2, tau2, cspeed2,
                ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                f20, f21, f22, f23, f24, f25, f26, f27, f28);
      break;
      case 8:
      convection_diffusion(mass1, tau1, cspeed1,
                  f10, f11, f12, f13, f14, f15, f16, f17, f18,
                  ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                  neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                  neq17, neq18);
      convection_diffusion(mass2, tau2, cspeed2,
                  f20, f21, f22, f23, f24, f25, f26, f27, f28,
                  ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                  neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                  neq27, neq28);
      break;
    }
  test_distribution_functions(ff10,ff11,ff12,ff13,ff14,ff15,ff16,
                              ff17,ff18);
  test_distribution_functions(ff20,ff21,ff22,ff23,ff24,ff25,ff26,
                              ff27,ff28);
  compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,ff16,ff17,ff18,
                       ff20,ff21,ff22,ff23,ff24,ff25,ff26,ff27,ff28);
  compute_equilibrium_distributions();
  switch(key_scheme)
    {
    case 0:
    case 10:
      first_upwind(mass1, tau1, cspeed1,
                   ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                   f10, f11, f12, f13, f14, f15, f16, f17, f18,
                   neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                   neq17, neq18);
```

```
        first_upwind(mass2, tau2, cspeed2,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                     neq27, neq28);
        break;
      case 1:
        second_upwind(mass1, tau1, cspeed1,
                      ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                      f10, f11, f12, f13, f14, f15, f16, f17, f18,
                      neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                      neq17, neq18);
        second_upwind(mass2, tau2, cspeed2,
                      ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                      f20, f21, f22, f23, f24, f25, f26, f27, f28,
                      neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                      neq27, neq28);
        break;
      case 2:
        first_centered(mass1, tau1, cspeed1,
                       ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                       f10, f11, f12, f13, f14, f15, f16, f17, f18,
                       neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                       neq17, neq18);
        first_centered(mass2, tau2, cspeed2,
                       ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                       f20, f21, f22, f23, f24, f25, f26, f27, f28,
                       neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                       neq27, neq28);
        break;
      case 3:
        lax_friedrichs(mass1, tau1, cspeed1,
                       ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                       f10, f11, f12, f13, f14, f15, f16, f17, f18,
                       neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                       neq17, neq18);
        lax_friedrichs(mass2, tau2, cspeed2,
                       ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                       f20, f21, f22, f23, f24, f25, f26, f27, f28,
                       neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                       neq27, neq28);
        break;
      case 4:
        lax_wendroff(mass1, tau1, cspeed1,
                       ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                       f10, f11, f12, f13, f14, f15, f16, f17, f18,
                       neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                       neq17, neq18);
        lax_wendroff(mass2, tau2, cspeed2,
                       ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                       f20, f21, f22, f23, f24, f25, f26, f27, f28,
                       neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                       neq27, neq28);
        break;
      case 5:
        leapfrog(mass1, tau1, cspeed1,
                       ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                       f10, f11, f12, f13, f14, f15, f16, f17, f18,
                       neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                       neq17, neq18);
        leapfrog(mass2, tau2, cspeed2,
                       ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                       f20, f21, f22, f23, f24, f25, f26, f27, f28,
                       neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                       neq27, neq28);
        break;
      case 6:
        key_point = 1;
        runge_kutta(key_point, mass1, tau1, cspeed1,
```



```
                    ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                    f10, f11, f12, f13, f14, f15, f16, f17, f18,
                    sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
                    neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                    neq17, neq18);
        runge_kutta(key_point, mass2, tau2, cspeed2,
                    ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                    f20, f21, f22, f23, f24, f25, f26, f27, f28,
                    sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
                    neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                    neq27, neq28);
        break;
      case 7:
        break;
      case 8:
        convection_diffusion(mass1, tau1, cspeed1,
                     ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                     f10, f11, f12, f13, f14, f15, f16, f17, f18,
                     neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                     neq17, neq18);
        convection_diffusion(mass2, tau2, cspeed2,
                     ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                     f20, f21, f22, f23, f24, f25, f26, f27, f28,
                     neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                     neq27, neq28);
        break;
      }
    autiter++;
    iter++;
    if(key_scheme < 6)
      {
        autiter++;
        iter++;
      }
  }

/*  Runge Kutta  or  lattice gas like  */
/*
while( autiter < niter_cycle )
  {
    store_distribution_functions();
    key_point = 0;
    test_distribution_functions(f10,f11,f12,f13,f14,f15,f16,f17,f18);
    test_distribution_functions(f20,f21,f22,f23,f24,f25,f26,f26,f28);
    compute_local_speeds(f10,f11,f12,f13,f14,f15,f16,f17,f18,
                         f20,f21,f22,f23,f24,f25,f26,f26,f27);
    compute_equilibrium_distributions();
    if(key_boundary > 4)
      {
        lglb_coll(mass1, tau1, cspeed1,
                  f10, f11, f12, f13, f14, f15, f16, f17, f18,
                  ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                  neq10, neq11, neq12, neq13, neq14, neq15, neq16,
                  neq17, neq18);
        lglb_coll(mass2, tau2, cspeed2,
                  f20, f21, f22, f23, f24, f25, f26, f27, f28,
                  ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                  neq20, neq21, neq22, neq23, neq24, neq25, neq26,
                  neq27, neq28);
        lglb_prop(mass1, tau1, cspeed1,
                  ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
                  f10, f11, f12, f13, f14, f15, f16, f17, f18);
        lglb_prop(mass2, tau2, cspeed2,
                  ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
                  f20, f21, f22, f23, f24, f25, f26, f27, f28);
      }
    else
      {
        runge_kutta(key_point, mass1, tau1, cspeed1,
```

```
        f10, f11, f12, f13, f14, f15, f16, f17, f18,
        ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
        sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
        neq10, neq11, neq12, neq13, neq14, neq15, neq16,
        neq17, neq18);
        runge_kutta(key_point, mass2, tau2, cspeed2,
        f20, f21, f22, f23, f24, f25, f26, f27, f28,
        ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
        sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
        neq20, neq21, neq22, neq23, neq24, neq25, neq26,
        neq27, neq28);
          key_point = 1;
          test_distribution_functions(ff10,ff11,ff12,ff13,ff14,ff15,ff16,
                                      ff17,ff18);
          test_distribution_functions(ff20,ff21,ff22,ff23,ff24,ff25,ff26,
                                      ff27,ff28);
          compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,ff16,ff17,ff18,
                               ff20,ff21,ff22,ff23,ff24,ff25,ff26,ff27,ff28);
          compute_equilibrium_distributions();
          runge_kutta(key_point, mass1, tau1, cspeed1,
          ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,
          f10, f11, f12, f13, f14, f15, f16, f17, f18,
          sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18,
          neq10, neq11, neq12, neq13, neq14, neq15, neq16,
          neq17, neq18);
          runge_kutta(key_point, mass2, tau2, cspeed2,
          ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28,
          f20, f21, f22, f23, f24, f25, f26, f27, f28,
          sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28,
          neq20, neq21, neq22, neq23, neq24, neq25, neq26,
          neq27, neq28);
        }
      autiter++;
      iter++;
    }
  */
}


void main(void)
{
  int isim, icycle;
  FILE *frez;

  dif9_input();
  for(isim=0; isim<nsim; isim++)
    {
      nnodes_x = a_nnodes_x[isim];
      nnodes_y = a_nnodes_y[isim];
      nnodes_all = a_nnodes_all[isim];
      lambda = a_lambda[isim];
      key_init = a_key_init[isim];
      key_boundary = a_key_boundary[isim];
      key_force = a_key_force[isim];
      key_scheme = a_key_scheme[isim];
      ncycles = a_ncycles[isim];
      niter_cycle = a_niter_cycle[isim];
      niter_init = a_niter_init[isim];
      length_x = a_length_x[isim];
      length_y = a_length_y[isim];
      delta_x = a_delta_x[isim];
      delta_y = a_delta_y[isim];
      delta_t = a_delta_t[isim];
      force_x = a_force_x[isim];
      force_y = a_force_y[isim];
      mass1 = a_mass1[isim];
      mass2 = a_mass2[isim];
      cspeed1 = a_cspeed1[isim];
```



```
      cspeed2 = a_cspeed2[isim];
      tau1 = a_tau1[isim];
      tau2 = a_tau2[isim];
      dcoef = a_dcoef[isim];
      nzero1left = a_nzero1left[isim];
      nzero2left = a_nzero2left[isim];
      nzero1right = a_nzero1right[isim];
      nzero2right = a_nzero2right[isim];

      cspeed12 = cspeed1 * cspeed1;
      cspeed22 = cspeed2 * cspeed2;
      cspeed1s = cspeed1 * sqrt((double) 2);
      cspeed2s = cspeed2 * sqrt((double) 2);
      cspeed1s2 = cspeed1s * cspeed1s;
      cspeed2s2 = cspeed2s * cspeed2s;

      build_names(isim);
      init_lattice_functions();
      init_ecx_nine();
      init_arrays_nine_square();
      init_arrays_nine_square_aux();
      getavec_square();

      iter = 0;
/*
      xv_new(f10,f11,f12,f13,f14,f15,f16,f17,f18,xv_name,1.0001);
      xv(f10,f11,f12,f13,f14,f15,f16,f17,f18,"XV");
*/
      test_ntot();
      dif9_profile();

      for(icycle=0; icycle<ncycles; icycle++)
        {
          dif9_automaton();
          /*
          xv_new(f10,f11,f12,f13,f14,f15,f16,f17,f18,xv_name,1.0001);
          */
          test_ntot();
          dif9_profile();
        }
      /*
      quiver();
      */
      free_lattice_functions();
    }
  free_nsim();
}
```

```
      nsim=   1
  nnodes_x=   250          nnodes_y= 5
    lambda=   1
   delta_x=   0.00001         delta_y=  0.00001      delta_t= 0.000000001
  key_init=   0         key_boundary=  2          key_force= 0 key_scheme= 4
   ncycles=  10          niter_cycle= 100          nter_init= 0
     mass1=   1.00               tau1=  0.00000001
     mass2=   0.90               tau2=  0.00000001
nzero1left= 1.00          nzero1right=  0.00
nzero2left= 0.00          nzero2right=  0.00
     dcoef= 0.00000000001
   force_x=   0.00000         force_y=  0.0000
```

Appendix C

sw code

```c
/*********************************************************************\
*                                                                     *
*  swhead.h                                                           *
*                                                                     *
*  Definition of global variables                                     *
*                                                                     *
\*********************************************************************/

#define NNX 1024
#define NNY 192
#define NNODES1 NNX * NNY + 1
#define NSIM 10


#ifdef MAIN_HEADER

  char sim_name[40], asim_name[NSIM][40];

  char job_name[]  = "sw.job",
       output_name[128], save_name[128], rez_name[128],
       profile_name[128], xv_name[128], wet_name[128], angle_name[128];

  double ecx[7], ecy[7], ecxx[7], ecxy[7], ecyx[7], ecyy[7];

  double f0[NNODES1],
         f1[NNODES1], f2[NNODES1], f3[NNODES1], f4[NNODES1],
         f5[NNODES1], f6[NNODES1];

  double ff0[NNODES1],
         ff1[NNODES1], ff2[NNODES1], ff3[NNODES1], ff4[NNODES1],
         ff5[NNODES1], ff6[NNODES1];

  double floc[NNODES1], fgradx[NNODES1], fgrady[NNODES1], flap[NNODES1];

  double edip[NNODES1], egradx[NNODES1], egrady[NNODES1];

  int val[NNODES1];

  int nv0[NNODES1],
      nv1[NNODES1], nv2[NNODES1], nv3[NNODES1], nv4[NNODES1],
      nv5[NNODES1], nv6[NNODES1];

  int boundary_mode[NNODES1];

  double rhomed, temp, temp_in, temp_fin, kappa, tau, tau3, ax, ay,
         mm, mx, my;
  double gradmiu1wall1, gradmiu2wall1, gradmiu1wall2, gradmiu2wall2;

  int key_status,
      key_init,
      key_interaction,
      key_interaction_ini,
      key_interaction_fin,
      key_boundary,
      key_boundary_ini,
      key_boundary_fin;

  int akey_status[NSIM], aniter_init[NSIM], ansav[NSIM],
      ancycles[NSIM], aniter_cycle[NSIM], akey_init[NSIM],
      akey_interaction_ini[NSIM], akey_interaction_fin[NSIM],
      akey_boundary_ini[NSIM], akey_boundary_fin[NSIM];

  double arhomed[NSIM], atemp_in[NSIM], atemp_fin[NSIM],
         akappa[NSIM], atau[NSIM], aax[NSIM], aay[NSIM],
         amx[NSIM], amy[NSIM],
         agradmiu1wall1[NSIM], agradmiu2wall1[NSIM],
         agradmiu1wall2[NSIM], agradmiu2wall2[NSIM];

  int av[101], as[101], achi[101], atime[101];
```



```c
  double ar1[101], aalpha1[101], aalphad1[101], atanalpha1[101],
         ar2[101], aalpha2[101], aalphad2[101], atanalpha2[101],
         acosalpha1[101], acosalpha2[101];

  int timecount;

  int nsim, niter_init, nsav, ncycles, niter_cycle, iter, niter;

  int key, scale;

  const double b = 6.000000000, b1 = 7.000000000,
         apsi = 9.0/49.0, bpsi = 2.0/21.0;

  const int nnodx = NNX, nnody = NNY, nnodes = NNX * NNY,
        nnodes1 = NNX * NNY +1, nnodesxy1 = NNX * (NNY - 1),
        nnodes_aux = NNX * NNY + 2 * (NNY + NNX + 1) + 1;

  const double dnnodx = (double) NNX, dnnody = (double) NNY;

  const double three_over_two = 3.00 / 2.00, nine_over_two = 9.00 / 2.00;

#else

  extern char sim_name[40], asim_name[NSIM][40];

  extern char job_name[],
         output_name[], save_name[], rez_name[],
         profile_name[], xv_name[], wet_name[], angle_name[];

  extern double ecx[], ecy[], ecxx[], ecxy[], ecyx[], ecyy[];

  extern double f0[], f1[], f2[], f3[], f4[], f5[], f6[];

  extern double ff0[], ff1[], ff2[], ff3[], ff4[], ff5[], ff6[];

  extern double floc[], fgradx[], fgrady[], flap[];

  extern double edip[], egradx[], egrady[];

  extern int val[];

  extern int nv0[], nv1[], nv2[], nv3[], nv4[], nv5[], nv6[];

  extern int boundary_mode[];

  extern double rhomed, temp, temp_in, temp_fin, kappa, tau, tau3, ax, ay,
                mm, mx, my;
  extern double gradmiu1wall1, gradmiu2wall1, gradmiu1wall2, gradmiu2wall2;

  extern int key_status, key_init,
             key_interaction, key_interaction_ini, key_interaction_fin,
             key_boundary, key_boundary_ini, key_boundary_fin;

  extern int akey_status[], aniter_init[], ansav[],
             ancycles[], aniter_cycle[], akey_init[],
             akey_interaction_ini[], akey_interaction_fin[],
             akey_boundary_ini[], akey_boundary_fin[];

  extern double arhomed[], atemp_in[], atemp_fin[], akappa[],
                atau[], aax[], aay[], amx[], amy[],
                agradmiu1wall1[], agradmiu2wall1[],
                agradmiu1wall2[], agradmiu2wall2[];

  extern int av[], as[], achi[], atime[];

  extern double ar1[], aalpha1[], aalphad1[], atanalpha1[],
                ar2[], aalpha2[], aalphad2[], atanalpha2[],
                acosalpha1[], acosalpha2[];
```

```
  extern int timecount;

  extern int nsim, niter_init, nsav, ncycles, niter_cycle, iter, niter;

  extern int key, scale;

  extern const double b, b1, apsi, bpsi;

  extern const int nnodx, nnody, nnodes, nnodes1, nnodesxy1,
                   nnodes_aux;

  extern const dnnodx, dnnody;

  extern const double three_over_two, nine_over_two;

#endif
```

```
/**********************************************************************\
*                                                                      *
*  swinout.c                                                           *
*                                                                      *
\**********************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include <string.h>
#include "swhead.h"

void sw_input(void)
{
  int i, j;
  FILE *fin;
  fin = fopen(job_name,"r");
  fscanf(fin,"%d\n",&nsim);
  if(nsim < 1)
    exit(1);
  for(i=0; i<nsim; i++)
    {
      fscanf(fin,"%d %s\n",&key_status,sim_name);
      fscanf(fin,"%d %d %d %d\n",&niter_init,&nsav,&ncycles,&niter_cycle);
      fscanf(fin,"%d %d %d %d %d\n",&key_init,
             &key_interaction_ini,&key_interaction_fin,
             &key_boundary_ini,&key_boundary_fin);
      fscanf(fin,"%lf %lf %lf %lf %lf\n",
             &rhomed,&temp_in,&temp_fin,&kappa,&tau);
      fscanf(fin,"%lf %lf %lf %lf\n",&ax,&ay,&mx,&my);
      fscanf(fin,"%lf %lf %lf %lf\n",&gradmiu1wall1,&gradmiu2wall1,
             &gradmiu1wall2,&gradmiu2wall2);
      akey_status[i] = key_status;
      j = 0;
      do
        {
          asim_name[i][j] = sim_name[j];
          j++;
        }
      while(sim_name[j]);
      aniter_init[i] = niter_init;
      ansav[i] = nsav;
      ancycles[i] = ncycles;
      aniter_cycle[i] = niter_cycle;
      akey_init[i] = key_init;
      akey_interaction_ini[i] = key_interaction_ini;
      akey_interaction_fin[i] = key_interaction_fin;
      akey_boundary_ini[i] = key_boundary_ini;
      akey_boundary_fin[i] = key_boundary_fin;
      arhomed[i] = rhomed;
      atemp_in[i] = temp_in;
      atemp_fin[i] = temp_fin;
      atau[i] = tau;
      akappa[i] = kappa;
      aax[i] = ax;
      aay[i] = ay;
      amx[i] = mx;
      amy[i] = my;
      agradmiu1wall1[i] = gradmiu1wall1;
      agradmiu2wall1[i] = gradmiu2wall1;
      agradmiu1wall2[i] = gradmiu1wall2;
      agradmiu2wall2[i] = gradmiu2wall2;
    }
}

void sw_output(void)
{
  int i;
  FILE *fout;
```



```
  fout = fopen("sw.out","w");
  fprintf(fout,"%d\n",nsim);
  for(i=0; i<nsim; i++)
    {
      fprintf(fout,"%d %s\n",akey_status[i],asim_name[i]);
      fprintf(fout,"%d %d %d %d\n",aniter_init[i],ansav[i],
              ancycles[i],aniter_cycle[i]);
      fprintf(fout,"%d %d %d %d %d\n",akey_init[i],
             akey_interaction_ini[i],akey_interaction_fin[i],
             akey_boundary_ini[i],akey_boundary_fin[i]);
      fprintf(fout,"%lf %lf %lf %lf %lf\n",arhomed[i],
              atemp_in[i],atemp_fin[i],atau[i]);
      fprintf(fout,"%lf %lf %lf %lf %lf\n",akappa[i],aax[i],aay[i],
              amx[i],amy[i]);
      fprintf(fout,"%lf %lf %lf %lf\n",agradmiu1wall1[i],agradmiu2wall1[i],
              agradmiu1wall2[i],agradmiu2wall2[i]);
    }
  fclose(fout);
}

void build_names(int isim)
{
  int irhomed,itemp_in,itemp_fin,itau,ikappa,iax,iay,imx,imy;
  int igradmiu1wall1,igradmiu2wall1,igradmiu1wall2,igradmiu2wall2;

  irhomed = (int) (arhomed[isim] * 100.00);
  itemp_in = (int) (atemp_in[isim] * 1000.00);
  itemp_fin = (int) (atemp_fin[isim] * 1000.00);
  itau = (int) (atau[isim] * 10.00);
  ikappa = (int) (akappa[isim]*10000.00);
  iax = (int) (aax[isim]*1000.00);
  iay = (int) (aay[isim]*10000000.00);
  imx = (int) (amx[isim]*1000.00);
  imy = (int) (amy[isim]*1000.00);
  igradmiu1wall1 = (int) (agradmiu1wall1[isim]*1.e+03);
  igradmiu2wall1 = (int) (agradmiu2wall1[isim]*1.e+03);
  igradmiu1wall2 = (int) (agradmiu1wall2[isim]*1.e+04);
  igradmiu2wall2 = (int) (agradmiu2wall2[isim]*1.e+04);
  sprintf(output_name,"%2.2d_%1d%1d_%3.3d_%3.3d_%3.3d_%4.4d_%2.2d_%3.3d_%8.8d_%3.3d_%3.3
d_%4.4d_%4.4d_%5.5d_%5.5d",
          akey_init[isim],akey_interaction_fin[isim],akey_boundary_fin[isim],
          irhomed,itemp_in,itemp_fin,ikappa,itau,
          iax,iay,imx,imy,igradmiu1wall1,igradmiu2wall1,
          igradmiu1wall2,igradmiu2wall2);
          sprintf(save_name,"S%s",output_name);
          sprintf(rez_name,"R%s",output_name);
          sprintf(profile_name,"P%s",output_name);
          sprintf(xv_name,"X_%s",output_name);
          sprintf(wet_name,"W_%s",output_name);
          sprintf(angle_name,"G_%s",output_name);
          /*
          printf("%s\n%s\n%s\n",save_name,rez_name,xv_name,angle_name);
          */
}

int read_status(void)
{
  FILE *fsav;
  if((fsav = fopen(save_name,"r")) == NULL);
  {
    return 1;
  }
/*
  if(
     (fread(&iter, sizeof iter, 1, fsav) != 1) ||
     (fread(f0, sizeof f0, 1, fsav) != 1) ||
     (fread(f1, sizeof f1, 1, fsav) != 1) ||
     (fread(f2, sizeof f2, 1, fsav) != 1) ||
     (fread(f3, sizeof f3, 1, fsav) != 1) ||
```

```
      (fread(f4, sizeof f4, 1, fsav) != 1) ||
      (fread(f5, sizeof f5, 1, fsav) != 1) ||
      (fread(f6, sizeof f6, 1, fsav) != 1) ||
      (fread(f7, sizeof f7, 1, fsav) != 1) ||
      (fread(f8, sizeof f8, 1, fsav) != 1)
      )
     {
      return 1;
     }
   if(
      (fread(g0, sizeof g0, 1, fsav) != 1) ||
      (fread(g1, sizeof g1, 1, fsav) != 1) ||
      (fread(g2, sizeof g2, 1, fsav) != 1) ||
      (fread(g3, sizeof g3, 1, fsav) != 1) ||
      (fread(g4, sizeof g4, 1, fsav) != 1) ||
      (fread(g5, sizeof g5, 1, fsav) != 1) ||
      (fread(g6, sizeof g6, 1, fsav) != 1) ||
      (fread(g7, sizeof g7, 1, fsav) != 1) ||
      (fread(g8, sizeof g8, 1, fsav) != 1)
      )
     {
      return 1;
     }
*/
  fclose(fsav);
  return 0;
}

int write_status(void)
{
  FILE *fsav;
  if((fsav = fopen(save_name,"w")) == NULL);
  {
    return 1;
  }
/*
  if(
      (fwrite(&iter, sizeof iter, 1, fsav) != 1) ||
      (fwrite(f0, sizeof f0, 1, fsav) != 1) ||
      (fwrite(f1, sizeof f1, 1, fsav) != 1) ||
      (fwrite(f2, sizeof f2, 1, fsav) != 1) ||
      (fwrite(f3, sizeof f3, 1, fsav) != 1) ||
      (fwrite(f4, sizeof f4, 1, fsav) != 1) ||
      (fwrite(f5, sizeof f5, 1, fsav) != 1) ||
      (fwrite(f6, sizeof f6, 1, fsav) != 1) ||
      (fwrite(f7, sizeof f7, 1, fsav) != 1) ||
      (fwrite(f8, sizeof f8, 1, fsav) != 1)
      )
     {
      return 1;
     }
  if(
      (fwrite(g0, sizeof (g0), 1, fsav) != 1) ||
      (fwrite(g1, sizeof g1, 1, fsav) != 1) ||
      (fwrite(g2, sizeof g2, 1, fsav) != 1) ||
      (fwrite(g3, sizeof g3, 1, fsav) != 1) ||
      (fwrite(g4, sizeof g4, 1, fsav) != 1) ||
      (fwrite(g5, sizeof g5, 1, fsav) != 1) ||
      (fwrite(g6, sizeof g6, 1, fsav) != 1) ||
      (fwrite(g7, sizeof g7, 1, fsav) != 1) ||
      (fwrite(g8, sizeof g8, 1, fsav) != 1)
      )
     {
      return 1;
     }
*/
  fclose(fsav);
  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "swhead.h"

void print_tavec(int kk, int nv[])
{
  int i,j,k;
  FILE *fout;
  fout = fopen(rez_name,"aw");
  fprintf(fout,"\nnv%1d\n",kk);
  k=0;
  for(j=1;j<=nnody;j++)
    {
      for(i=1;i<=nnodx;i++)
        {
 k++;
          fprintf(fout,"%3d ",nv[(j-1)*nnodx+i]);
        }
      fprintf(fout,"\n");
    }
  fclose(fout);
}

void getavec_hex(void)
{
  int i, j, k, i1;
  int nn[nnodes+1+2*(nnody+nnodx+1)+1+nnodes];
  int in = nnodx+1, n0, n1, n2, n3, n4, n5, n6, nnl, ip;
  for(i=1;i<=nnody;i++)
    {
      in++;
      nn[in]=i*nnodx;
      for(j=1;j<=nnodx;j++)
        {
          in++;
          nn[in]=(i-1)*nnodx+j;
        }
      in++;
      nn[in]=(i-1)*nnodx+1;
    }
  in=0;
  for(i=1;i<=nnodx;i++)
    {
      in++;
      nn[in]=(nnody-1)*nnodx+i;
    }
  in++;
  nn[in]=nn[1];
  in=(nnodx+1)+nnody*(nnodx+2);
  for(i=1;i<=nnodx+1;i++)
    {
      in++;
      nn[in]=nn[nnodx+1+i];
    }
  in++;
  nn[in]=1;
  n0=0;
  for(i=1;i<=nnody;i++)
    {
      i1=i*(nnodx+2);
      for(j=1;j<=nnodx;j++)
        {
          nnl=i1+j;
          ip=i%2;
          n0++;
          n1=nnl+1;
```



```c
          n2=nnl-(nnodx+2-ip);
          n3=n2-1;
          n4=nnl-1;
          n5=nnl+(nnodx+1+ip);
          n6=n5+1;
          nv0[n0]=n0;
          nv1[n0]=nn[n1];
          nv2[n0]=nn[n2];
          nv3[n0]=nn[n3];
          nv4[n0]=nn[n4];
          nv5[n0]=nn[n5];
          nv6[n0]=nn[n6];
        }
    }
/*
  print_tavec(0,nv0);
  print_tavec(1,nv1);
  print_tavec(2,nv2);
  print_tavec(3,nv3);
  print_tavec(4,nv4);
  print_tavec(5,nv5);
  print_tavec(6,nv6);
*/
}


void init_arrays_hex()
{
  FILE *fsav, *frez;
  int i,j,k,ic=nnodx/2,jc=nnody/2, scala = 14, key_rho, layer_height;
  double rand_coef = 0.1, rho_high, rho_low;
  double x,y,raza,raza_max=nnody*sqrt(3.)/4.,xc,yc;
  iter = 0;
  frez=fopen(rez_name,"a");
  fprintf(frez,"\n---  init_arrays ---\n");
  ecx[1]=1.00;
  ecy[1]=0.00;
  ecx[2]=0.50;
  ecy[2]=sqrt(3.00)/2.0;
  ecx[3]=-ecx[2];
  ecy[3]=ecy[2];
  ecx[4]=-ecx[1];
  ecy[4]=0.00;
  ecx[5]=ecx[3];
  ecy[5]=-ecy[3];
  ecx[6]=ecx[2];
  ecy[6]=ecy[5];
  for( i=1; i <= 6; i++)
    {
      ecxx[i] = ecx[i]*ecx[i];
      ecxy[i] = ecx[i]*ecy[i];
      ecyx[i] = ecy[i]*ecx[i];
      ecyy[i] = ecy[i]*ecy[i];
    }

  for( i=1; i <= 6; i++)
    {
      fprintf(frez,"i,ecx[i],ecy[i]\n%3d %25.16e %25.16e\n",i,ecx[i],ecy[i]);
      fprintf(frez,"i,ecxx[i],ecxy[i],ecyx[i],ecyy[i]\n%3d %25.16e %25.16e %25.16e %25.1
6e\n",
i,ecxx[i],ecxy[i],ecyx[i],ecyy[i]);
    }

  switch (key_init)
    {
    case 0: /* init condens */
      for( i = 1; i <= nnodes; i++)
        {
          f0[i] = (rhomed / b1 )*(1. +rand_coef *
```

```
                                   ( ((double) rand() /
                                          (double) RAND_MAX) - 0.5));
             }
        break;
      case 1: /* init full square */
      case 2: /* init void square */

        rho_high = 1.1 * rho_high;
        k=0;
        for (j=1; j<=nnody; j++)
          for (i=1; i<=nnodx; i++)
            {
              k++;
              f0[k] = 1.000/( b1 ) * (1.+rand_coef *
                                     ( ((double) rand() /(double) RAND_MAX) - 0.5));
            }
        for (j=nnody/4-2; j<=(3*nnody)/4+2; j++)
          for(i=nnodx/4-2; i<=(3*nnodx)/4+2; i++)
            {
              k=nnody*i+j;
              f0[k] = 1.000/( b1 )*
                (1.+rand_coef*( ((double) rand() /
                                 (double) RAND_MAX) - 0.5));
            }
        for (j=nnody/4; j<=(3*nnody)/4; j++)
          for(i=nnodx/4; i<=(3*nnodx)/4; i++)
            {
              k=nnody*i+j;
              f0[k] = 1.000/( b1 )*
                (1.+rand_coef*( ((double) rand() /
                                     (double) RAND_MAX) - 0.5));
            }
        break;
      case 3: /* init drop   */
      case 4: /* init bubble */
        {
          raza=((double) scala)*raza_max/16.;
          xc=((double) nnodx)/2.+((double)(nnody-(nnody/2)*2))/2.;
          yc=((double) nnody)*sqrt(3.)/4.;
          k=0;
          for(j=1; j <= nnody; j++)
            {
              y=sqrt(3.)*((double) j)/2.;
              for(i=1; i <= nnodx; i++)
                {
                  x=((double) i)+((double) (j-(j/2)*2))/2.;
                  k++;
                  if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=(raza+2.)*(raza+2.))
                    key_rho=2;
                  else
                    key_rho=3;
                  if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza)
                    key_rho=1;
                  switch (key_rho)
                    {
                    case 1:
                      f0[k] = rhomed * 1.35/( b1 )*
                        (1.+rand_coef*( ((double) rand() /
                                         (double) RAND_MAX) - 0.5));
                      break;
                    case 2:
                      f0[k] = rhomed/( b1 )*
                        (1.+rand_coef*( ((double) rand() /
                                         (double) RAND_MAX) - 0.5));
                      break;
                    case 3:
                      f0[k] = rhomed * 0.65/( b1 )*
                        (1.+rand_coef*( ((double) rand() /
                                         (double) RAND_MAX) - 0.5));
```



```
                      break;
                    }
/*
                  switch (key_rho)
                    {
                    case 1:
                      f0[k] = 1.000/( b1 )*
                        (1.+rand_coef*( ((double) rand() /
                                         (double) RAND_MAX) - 0.5));
                      break;
                    case 2:
                      f0[k] = 1.000/( b1 )*
                        (1.+rand_coef*( ((double) rand() /
                                         (double) RAND_MAX) - 0.5));
                      break;
                    case 3:
                      f0[k] = 1.000/( b1 )*
                        (1.+rand_coef*( ((double) rand() /
                                         (double) RAND_MAX) - 0.5));
                      break;
                    }
*/
                }
            }
        }
        break;
      case 5: /* init banda orizontala */
        for ( j = 1; j <= nnody; j++)
          {
            k=(j-1)*nnodx;
            for(i=1; i<=nnodx; i++)
              {
                k++;
                f0[k] = rhomed/( b1 )*
                  (1.+rand_coef*( ((double) rand() /
                                   (double) RAND_MAX) - 0.5));
                if(j<nnody/2)
                  {
                    f0[k] *= 1.2;
                  }
                else
                  {
                    f0[k] *= 0.8;
                  }
              }
          }
        break;
      case 7: /* init half drop */
      case 8: /* init bubble */
        {
          raza=2.00 * ((double) scala)*raza_max/16.;
          xc=((double) nnodx)/2.+((double)(nnody-(nnody/2)*2))/2.;
          yc=((double) nnody) * sqrt(3.00) / 2.00;
          k=0;
          for(j=1; j <= nnody; j++)
            {
              y=sqrt(3.)*((double) j)/2.;
              for(i=1; i <= nnodx; i++)
                {
                  x=((double) i)+((double) (j-(j/2)*2))/2.;
                  k++;
                  if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=(raza+2.)*(raza+2.))
                    key_rho=2;
                  else
                    key_rho=3;
                  if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza)
                    key_rho=1;
                  switch (key_rho)
                    {
```

```
                case 1:
                  f0[k] = rhomed * 1.35/( b1 )*
                    (1.+rand_coef*( ((double) rand() /
                                     (double) RAND_MAX) - 0.5));
                  break;
                case 2:
                  f0[k] = rhomed/( b1 )*
                    (1.+rand_coef*( ((double) rand() /
                                     (double) RAND_MAX) - 0.5));
                  break;
                case 3:
                  f0[k] = rhomed * 0.65/( b1 )*
                    (1.+rand_coef*( ((double) rand() /
                                     (double) RAND_MAX) - 0.5));
                  break;
                }
            }
        }
    }
    break;


  case 6: /* init banda verticala */
    /*
       for ( j = 1; j <= NNLAT; j++)
       {
       k=(j-1)*NNLUN;
       for(i=1; i<=NNLUN; i++)
       {
       k++;
       if( ( i >= (NNLUN-lat_banda)/2 -2 ) &&
       ( j <= (NNLUN+lat_banda)/2 +2 ) )
       f0[k] = rho_med/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       else
       f0[k] = rho_low/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       if( ( j >= (NNLUN-lat_banda)/2  ) &&
       ( j <= (NNLUN+lat_banda)/2  ) )
       f0[k] = rho_high/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       }
       }
       break;
       */
  case 9:
    /*
       layer_height = NNLAT * (rho_low / (rho_high+rho_low));
       if (NNLAT == 200)
       layer_height = layer_height / 2;
       if (key_init == 9)
       layer_height = NNLAT - layer_height;
       k = 0;
       for( j=1; j<= layer_height-3; j++)
       for(i=1; i<=NNLUN; i++)
       {
       k++;
       f0[k] = rho_low/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       }
       for( j=layer_height-2; j<= layer_height-1; j++)
       for(i=1; i<=NNLUN; i++)
       {
       k++;
       f0[k] = (rho_med+rho_low)/( 2.*b1 )*
```



```
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       }
       fcr( j=layer_height; j<= layer_height+1; j++)
       for(i=1; i<=NNLUN; i++)
       {
       k++;
       f0[k] = rho_med/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       }
       for( j=layer_height+2; j<= layer_height+3; j++)
       for(i=1; i<=NNLUN; i++)
       {
       k++;
       f0[k] = (rho_med+rho_high)/( 2.*b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       }
       for( j=layer_height+4; j<= NNLAT; j++)
       for(i=1; i<=NNLUN; i++)
       {
       k++;
       f0[k] = rho_high/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       }
       */
    break;
  case 10: /* init drop + layer  */
    /*
       {
       fprintf(frez,"rho_high = %f --- rho_low = %f\n",rho_high,rho_low);
       raza=((double) scala)*raza_max/16.;
       xc=((double) NNLUN)/2.+((double) (NNLAT-(NNLAT/2)*2))/2.;
       yc=((double) NNLAT)*sqrt(3.)/4.;
       k=0;
       for(j=1; j <= NNLAT; j++)
       {
       y=sqrt(3.)*((double) j)/2.;
       for(i=1; i <= NNLUN; i++)
       {
       x=((double) i)+((double) (j-(j/2)*2))/2.;
       k++;
       if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=(raza+2.)*(raza+2.))
       key_rho=2;
       else
       key_rho=3;
       if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza)
       key_rho=1;
       switch (key_rho)
       {
       case 1:
       f0[k] = rho_high/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       break;
       case 2:
       f0[k] = rho_med/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       break;
       case 3:
       f0[k] = rho_low/( b1 )*
       (1.+rand_coef*( ((double) rand() /
       (double) RAND_MAX) - 0.5));
       break;
       }
       }
```

```c
        }
        for(j=NNLAT-3; j<=NNLAT-2; j++)
        for(i=1; i<= NNLUN; i++)
        {
        k = j*NNLUN + (i-1);
        f0[k] = rho_med/( b1 )*
        (1.+rand_coef*( ((double) rand() /
        (double) RAND_MAX) - 0.5));
        }
        for(j=NNLAT-1; j <= NNLAT; j++)
        for(i=1; i<= NNLUN; i++)
        {
                k = j + (i-1);
                f0[k] = rho_high/( b1 )*
                (1.+rand_coef*( ((double) rand() /
                (double) RAND_MAX) - 0.5));
                printf("%d %d %d %f\n",j,i,k,f0[k]);
                }
         }
         */
      break;
    }

  for( i=1; i <= nnodes; i++)
    {
      f1[i] = f0[i];
      f2[i] = f0[i];
      f3[i] = f0[i];
      f4[i] = f0[i];
      f5[i] = f0[i];
      f6[i] = f0[i];
    }
  fclose(frez);
}

void init_arrays_hex_aux ()
{
  FILE *frez;
  int i,j,k;
  int nnl1 = (nnody-1)*nnodx;

  switch(key_boundary)
    {
    case 0:
      for ( i = 1; i<= nnodes; i++)
        boundary_mode[i] = 0;
      break;
    case 1:
      for ( i = 1; i<= nnodx; i++)
        boundary_mode[i] = 1;
      for ( i = nnodx+1; i<=nnl1; i++)
        boundary_mode[i] = 0;
      for ( i = nnl1+1; i<= nnodes; i++)
        boundary_mode[i] = 2;
      break;
/*
    case 3:
      for ( i = 1; i<= NNL1; i++)
        boundary_mode[i] = 0;
      for ( i = NNL1+1; i<= nnodes; i++)
        boundary_mode[i] = 2;
      break;
    case 2:
      for ( i = 2; i<= nnodx-1; i++)
        boundary_mode[i] = 1;
      for ( i = nnodx+1; i<=NNL1; i++)
        boundary_mode[i] = 0;
      for ( i = NNL1+2; i<= nnodes-1; i++)
        boundary_mode[i] = 2;
```



```c
      for (j=2; j<=NNLAT-1; j++)
        if ( j % 2 )
          {
            boundary_mode[(j-1)*NNLUN+1] = 5;
            boundary_mode[j*NNLUN] = 6;
          }
        else
          {
            boundary_mode[(j-1)*nnodx+1] = 3;
            boundary_mode[j*nnodx] = 4;
          }
      boundary_mode[1] = 7;
      boundary_mode[nnodx] = 8;
      boundary_mode[NNL1+1] = 9;
      boundary_mode[nnodes] = 10;
      break;
*/
    }


}
```

```c
#include <stdio.h>
#include <math.h>
#include "swhead.h"

void xv(double n0[], double n1[], double n2[], double n3[],
        double n4[], double n5[], double n6[], char arg_name[])
{
  FILE *fxv;
  char xv_name[128];
  int i,k,val;
  sprintf(xv_name,"%s.%05d",arg_name,iter);
  fxv = fopen(xv_name,"wt");
  fprintf(fxv,"P2\n%3d%4d\n63\n",nnodx,nnody);
  i=0;
  for(k=1; k<=nnodes; k++)
    {
      i++;
      val = floor((n0[k]+n1[k]+n2[k]+n3[k]+
                   n4[k]+n5[k]+n6[k])*63.0/7.00);
      if(val > 63)
        val = 63;
      if(val < 0)
        val = 0;
/*      val = 63 - val;   */
      fprintf(fxv,"%3d",val);
      if(i == 15)
        {
          i=0;
          fprintf(fxv,"\n");
        }
    }
  if(i)
    fprintf(fxv,"\n");
  fclose(fxv);
/*
  sprintf(xv_name,"%s.%05dP",arg_name,iter);
  fxv = fopen(xv_name,"wt");
  fprintf(fxv,"P2\n%3d%4d\n63\n",nnlun,nnlat);
  i=0;
  for(k=1; k<=nnod; k++)
    {
      i++;
      val = ( floor((f1[k]*ecx[1]+f2[k]*ecx[2]+f3[k]*ecx[3]+
                     f4[k]*ecx[4]+f5[k]*ecx[5]+f6[k]*ecx[6])*1000.));

      printf("%25.16f\n",(f1[k]*ecx[1]+f2[k]*ecx[2]+f3[k]*ecx[3]+
                          f4[k]*ecx[4]+f5[k]*ecx[5]+f6[k]*ecx[6]));
      printf("%d\n",val);

      if(val > 63)
        val = 63;
      if(val < 0)
        val = 0;
      val = 63 - val;

      fprintf(fxv,"%3d",val);
      if(i == 15)
        {
          i=0;
          fprintf(fxv,"\n");
        }
    }
  if(i)
    fprintf(fxv,"\n");
  fclose(fxv);
*/
}

void profile(double n0[], double n1[], double n2[], double n3[],
```



```c
        double n4[], double n5[], double n6[], char arg_name[])
{
  FILE *fxv;
  char xv_name[128];
  int i,k;
  double val;
  sprintf(xv_name,"%s.%05d",arg_name,iter);
  fxv = fopen(xv_name,"wt");
  for(i=1; i<=nnody; i++)
    {
      k=(i-1)*nnodx+nnodx/2;
      val = n0[k]+n1[k]+n2[k]+n3[k]+n4[k]+n5[k]+n6[k];
      fprintf(fxv,"%3d %lf\n",i,val);
    }
  fclose(fxv);
}

void minko(void)
{
  FILE *fwet, *falpha;
  int pix[NNODES1];
  double rholoc, s, v, coef, alpha1, alpha2, a, b, c, fa, fb, fc;
  double height, width, galpha, gtan, gcos;
  int ih, ihminus1, ihminus2, ihplus1, ihplus2, iw;
  int i,j,k;
  int iv, is;

  timecount++;
  iv = 0;

  for(i=1; i<=nnodes; i++)
    {
      rholoc = f0[i] + f1[i] + f2[i] + f3[i] + f4[i] + f5[i] + f6[i];
      if(rholoc > 3.5)
        {
          pix[i] = 1;
          iv ++;
        }
      else
        {
          pix[i] = 0;
        }
    }

  is = 0;
  for(i=1; i<=nnodes; i++)
    {
      if(pix[i])
        {
          if(!pix[nv1[i]])
            {
              is++;
            }
          if(!pix[nv4[i]])
            {
              is++;
            }
          switch(boundary_mode[i])
            {
            case 0:
              if(!pix[nv2[i]])
                {
                  is++;
                }
              if(!pix[nv3[i]])
                {
                  is++;
                }
              if(!pix[nv5[i]])
```

```
                {
                  is++;
                }
              if(!pix[nv6[i]])
                {
                  is++;
                }
              break;
            case 1:
              if(!pix[nv5[i]])
                {
                  is++;
                }
              if(!pix[nv6[i]])
                {
                  is++;
                }
              break;
            case 2:
              if(!pix[nv2[i]])
                {
                  is++;
                }
              if(!pix[nv3[i]])
                {
                  is++;
                }
              break;
            }
        }
    }

  is /= 2;
  v = (double) iv;
  v *= (sqrt(3.00) / 2.00);
  s = (double) is;
  coef = 4.00 * v / (s * s);
  a = 0.00000001;
  b = 3.14159 / 2.00;
  fa = 1.00 / a - sin(2.00 * a) / (2.00 *a *a) - coef;
  fb = 1.00 / b - sin(2.00 * b) / (2.00 *b *b) - coef;
  do
    {
      c = (a+b)/2.00;
      fc = 1.00 / c - sin(2.00 * c) / (2.00 *c *c) - coef;
/*
      printf("%lf %lf %lf %lf %lf %lf\n",a,b,c,fa,fb,fc);
*/
      if(fc)
        {
          if(fa*fc > 0.00)
            {
              a = c;
              fa = fc;
            }
          else
            {
              b = c;
              fb = fc;
            }
          alpha1 = (b+a) / 2.00;
        }
      else
        {
          alpha1 = c;
        }
    }
  while((b-a) > 0.001);
```



```
  a = 3.14159 / 2.00;
  b = 3.14159;
  fa = 1.00 / a - sin(2.00 * a) / (2.00 *a *a) - coef;
  fb = 1.00 / b - sin(2.00 * b) / (2.00 *b *b) - coef;
  do
    {
      c = (a+b)/2.00;
      fc = 1.00 / c - sin(2.00 * c) / (2.00 *c *c) - coef;
/*
      printf("%lf %lf %lf %lf %lf %lf\n",a,b,c,fa,fb,fc);
*/
      if(fc)
        {
          if(fa*fc > 0.00)
            {
              a = c;
              fa = fc;
            }
          else
            {
              b = c;
              fb = fc;
            }
          alpha2 = (b+a) / 2.00;
        }
      else
        {
          alpha2 = c;
        }
    }
  while((b-a) > 0.001);

  ih = 0;
  ihminus1 = ihminus2 = 0;
  ihplus1 = ihplus2 = 0;
  for(j=1; j<=nnody; j++)
    {
      k = (j-1)*nnodx + nnodx/2;
      if(!pix[k-1])
        ihminus1 = j;
      if(!pix[k-2])
        ihminus2 = j;
      if(!pix[k+1])
        ihplus1 = j;
      if(!pix[k+2])
        ihplus2 = j;
      if(!pix[k])
        ih = j;
      /*
      printf("j=%d  k=%d  pix=%d  ih=%d\n",j,k,pix,ih);
      */
    }
  if(ihminus1<ih)
    ih = ihminus1;
  if(ihminus2<ih)
    ih = ihminus2;
  if(ihplus1<ih)
    ih = ihplus1;
  if(ihplus2<ih)
    ih = ihplus2;
  height = ((double) (nnody - ih)) * sqrt(3.000) / 2.000;

  iw = 0;
  for(i=1; i<=nnodx; i++)
    {
      k = (nnody-1)*nnodx + i;
      if(pix[k])
        iw++;
      /*
```

```
    printf("i=%d  k=%d  pix=%d  iw=%d\n",i,k,pix,iw);
    */
  }
  width = ((double) iw);

  if(width)
    gtan = 2.000 * height / width;
  else
    gtan = 0.000;
  galpha = 2.000 * atan(gtan);
  gcos = cos(galpha);

  av[timecount] = iv;
  as[timecount] = is;
  ar1[timecount] = s / (2.00 * alpha1);
  ar2[timecount] = s / (2.00 * alpha2);
  aalpha1[timecount] = alpha1;
  aalpha2[timecount] = alpha2;
  aalphad1[timecount] = aalpha1[timecount] * 180.00 / 3.14159;
  aalphad2[timecount] = aalpha2[timecount] * 180.00 / 3.14159;
  atanalpha1[timecount] = tan(aalpha1[timecount]);
  atanalpha2[timecount] = tan(aalpha2[timecount]);
  acosalpha1[timecount] = cos(aalpha1[timecount]);
  acosalpha2[timecount] = cos(aalpha2[timecount]);
  atime[timecount] = iter;

  if(iter)
    {
      fwet = fopen(wet_name,"aw");
      falpha = fopen(angle_name,"aw");
    }
  else
    {
      fwet = fopen(wet_name,"w");
      falpha = fopen(angle_name,"w");
    }
  fprintf(fwet,"%d %d %d %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf\n",
          atime[timecount], av[timecount], as[timecount],
          ar1[timecount], ar2[timecount],
          aalpha1[timecount], aalpha2[timecount],
          aalphad1[timecount], aalphad2[timecount],
          atanalpha1[timecount], atanalpha2[timecount],
          acosalpha1[timecount], acosalpha2[timecount] );
  fprintf(falpha,"%d  %d  %d  %lf  %lf  %e  %e  %lf  %lf\n",
          iter,ih,iw,height,width,gtan,galpha,galpha*180.000/3.14159,gcos);
  fclose(fwet);
  fclose(falpha);
}

double local_gradient ( double *ec, double n1, double n2, double n3,
                        double n4, double n5, double n6 )
{
  return ( ec[1]*n1 + ec[2]*n2 + ec[3]*n3 +
           ec[4]*n4 + ec[5]*n5 + ec[6]*n6 ) / 3.;
}

void gradient(double n[], double *gradx, double *grady)
{
  int i;
  double doin, sq3 = sqrt(3.);
  double sq32 = sq3*2.;
  for(i=1; i<=nnodes; i++)
    switch(boundary_mode[i])
      {
      case 0:
        gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*n[nv2[i]]+
                    ecx[3]*n[nv3[i]]+ecx[4]*n[nv4[i]]+
                    ecx[5]*n[nv5[i]]+ecx[6]*n[nv6[i]])/3.;
        grady[i] = (ecy[1]*n[nv1[i]]+ecy[2]*n[nv2[i]]+
```



```
                    ecy[3]*n[nv3[i]]+ecy[4]*n[nv4[i]]+
                    ecy[5]*n[nv5[i]]+ecy[6]*n[nv6[i]])/3.;
        break;
      case 1:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*(doin-n[nv5[i]])+
                    ecx[3]*(doin-n[nv6[i]])+ecx[4]*n[nv4[i]]+
                    ecx[5]*n[nv5[i]]+ecx[6]*n[nv6[i]])/3.;
        grady[i] = (ecy[1]*n[nv1[i]]+ecy[2]*(doin-n[nv5[i]])+
                    ecy[3]*(doin-n[nv6[i]])+ecy[4]*n[nv4[i]]+
                    ecy[5]*n[nv5[i]]+ecy[6]*n[nv6[i]])/3.;
        break;
      case 2:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*n[nv2[i]]+
                    ecx[3]*n[nv3[i]]+ecx[4]*n[nv4[i]]+
                    ecx[5]*(doin-n[nv2[i]])+ecx[6]*(doin-n[nv3[i]]))/3.;
        grady[i] = (ecy[1]*n[nv1[i]]+ecy[2]*n[nv2[i]]+
                    ecy[3]*n[nv3[i]]+ecy[4]*n[nv4[i]]+
                    ecy[5]*(doin-n[nv2[i]])+ecy[6]*(doin-n[nv3[i]]))/3.;
        break;
      case 3:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*n[nv2[i]]+
                    ecx[3]*(doin-n[nv6[i]])+ecx[4]*(doin-n[nv1[i]])+
                    ecx[5]*(doin-n[nv2[i]])+ecx[6]*n[nv6[i]])/3.;
        grady[i] = (ecy[1]*n[nv1[i]]+ecy[2]*n[nv2[i]]+
                    ecy[3]*(doin-n[nv6[i]])+ecy[4]*(doin-n[nv1[i]])+
                    ecy[5]*(doin-n[nv2[i]])+ecy[6]*n[nv6[i]])/3.;
        break;
      case 4:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*(doin-n[nv4[i]])+ecx[2]*n[nv2[i]]+
                    ecx[3]*n[nv3[i]]+ecx[4]*n[nv4[i]]+
                    ecx[5]*n[nv5[i]]+ecx[6]*n[nv6[i]])/3.;
        grady[i] = (ecy[1]*(doin-n[nv4[i]])+ecy[2]*n[nv2[i]]+
                    ecy[3]*n[nv3[i]]+ecy[4]*n[nv4[i]]+
                    ecy[5]*n[nv5[i]]+ecy[6]*n[nv6[i]])/3.;
        break;
      case 5:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*n[nv2[i]]+
                    ecx[3]*n[nv3[i]]+ecx[4]*(doin-n[nv1[i]])+
                    ecx[5]*n[nv5[i]]+ecx[6]*n[nv6[i]])/3.;
        grady[i] = (ecy[1]*n[nv1[i]]+ecy[2]*n[nv2[i]]+
                    ecy[3]*n[nv3[i]]+ecy[4]*(doin-n[nv1[i]])+
                    ecy[5]*n[nv5[i]]+ecy[6]*n[nv6[i]])/3.;
        break;
      case 6:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*(doin-n[nv4[i]])+ecx[2]*(doin-n[nv5[i]])+
                    ecx[3]*n[nv3[i]]+ecx[4]*n[nv4[i]]+
                    ecx[5]*n[nv5[i]]+ecx[6]*(doin-n[nv3[i]]))/3.;
        grady[i] = (ecy[1]*(doin-n[nv4[i]])+ecy[2]*(doin-n[nv5[i]])+
                    ecy[3]*n[nv3[i]]+ecy[4]*n[nv4[i]]+
                    ecy[5]*n[nv5[i]]+ecy[6]*(doin-n[nv6[i]]))/3.;
        break;
      case 7:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*(doin-n[nv5[i]])+
                    ecx[3]*(doin-n[nv6[i]])+ecx[4]*(doin-n[nv1[i]])+
                    ecx[5]*n[nv5[i]]+ecx[6]*n[nv6[i]])/3.;
        grady[i] = (ecy[1]*n[nv1[i]]+ecy[2]*(doin-n[nv5[i]])+
                    ecy[3]*(doin-n[nv6[i]])+ecy[4]*(doin-n[nv1[i]])+
                    ecy[5]*n[nv5[i]]+ecy[6]*n[nv6[i]])/3.;
        break;
      case 8:
        doin = 2.*n[i];
        gradx[i] = (ecx[1]*(doin-n[nv4[i]])+ecx[2]*(doin-n[nv5[i]])+
```

```
                  ecx[4]*n[nv4[i]]+ecx[5]*n[nv5[i]])/3.;
      grady[i] = (ecy[1]*(doin-n[nv4[i]])+ecy[2]*(doin-n[nv5[i]])+
                  ecy[4]*n[nv4[i]]+ecy[5]*n[nv5[i]])/3.;
      break;
    case 9:
      doin = 2.*n[i];
      gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*n[nv2[i]]+
                  ecx[4]*(doin-n[nv1[i]])+ecx[5]*(doin-n[nv2[i]]))/3.;
      grady[i] = (ecy[1]*n[nv1[i]]+ecy[2]*n[nv2[i]]+
                  ecy[4]*(doin-n[nv1[i]])+ecy[5]*(doin-n[nv2[i]]))/3.;
      break;
    case 10:
      doin = 2.*n[i];
      gradx[i] = (ecx[1]*(doin-n[nv4[i]])+ecx[2]*n[nv2[i]]+
                  ecx[3]*n[nv3[i]]+ecx[4]*n[nv4[i]]+
                  ecx[5]*(doin-n[nv2[i]])+ecx[6]*(doin-n[nv3[i]]))/3.;
      grady[i] = (ecy[1]*(doin-n[nv4[i]])+ecy[2]*n[nv2[i]]+
                  ecy[3]*n[nv3[i]]+ecy[4]*n[nv4[i]]+
                  ecy[5]*(doin-n[nv2[i]])+ecy[6]*(doin-n[nv3[i]]))/3.;
      break;
    }
}

void laplacian(double n[], double *nlap)
{
  int i;
  double doipetrei = 2./3.;
  for(i=1; i<=nnodes; i++)
    switch(boundary_mode[i])
    {
    case 0:
      nlap[i] = doipetrei*(n[nv1[i]]+n[nv2[i]]+n[nv3[i]]+
                           n[nv4[i]]+n[nv5[i]]+n[nv6[i]] -6.*n[i]);
      break;
    case 1:
      nlap[i] = doipetrei*(n[nv1[i]]+n[nv4[i]]-2.*n[i]);
      break;
    case 2:
      nlap[i] = doipetrei*(n[nv1[i]]+n[nv4[i]]-2.*n[i]);
      break;
    case 3:
      nlap[i] = 0.;
      break;
    case 4:
      nlap[i] = doipetrei*(n[nv2[i]]+n[nv3[i]]+
                           n[nv5[i]]+n[nv6[i]]-4.*n[i]);
      break;
    case 5:
      nlap[i] = doipetrei*(n[nv2[i]]+n[nv3[i]]+
                           n[nv5[i]]+n[nv6[i]]-4.*n[i]);
      break;
    case 6:
      nlap[i] = 0.;
      break;
    case 7:
      nlap[i] = 0.;
      break;
    case 8:
      nlap[i] = 0.;
      break;
    case 9:
      nlap[i] = 0.;
      break;
    case 10:
      nlap[i] = 0.;
      break;
    }
}
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#include "swhead.h"

double rotot(double f0[], double f1[], double f2[], double f3[],
             double f4[], double f5[], double f6[], unsigned int nnod );
double local_gradient(double *ec, double n1, double n2, double n3,
                      double n4, double n5, double n6);
void gradient(double nloc[], double nx[], double ny[]);
void laplacian(double nloc[], double nlap[]);



void automaton_aux(double nf0[], double nf1[], double nf2[], double nf3[],
                   double nf4[], double nf5[], double nf6[],
                   double nff0[], double nff1[], double nff2[], double nff3[],
                   double nff4[], double nff5[], double nff6[])
{
  FILE *frez;
  int i, j, k;
  double uxloc, uyloc, uxxloc, uxyloc, uyyloc, uloc2;
  double cazero, ca, cb, cczero, cc, cd, cgxx, cgxy, xgyx, cgyy;
  double feq[7];
  double fx, fy;

  for(i=1; i<=nnodes; i++)
    {

      if(nf0[i] < 0.00)
        {
          printf("i=%d nf0=%lf\n",i,nf0[i]);
        }
      if(nf1[i] < 0.00)
        {
          printf("i=%d nf1=%lf\n",i,nf1[i]);
        }
      if(nf1[i] < 0.00)
        {
          printf("i=%d nf2=%lf\n",i,nf2[i]);
        }
      if(nf3[i] < 0.00)
        {
          printf("i=%d nf3=%lf\n",i,nf3[i]);
        }
      if(nf4[i] < 0.00)
        {
          printf("i=%d nf4=%lf\n",i,nf4[i]);
        }
      if(nf5[i] < 0.00)
        {
          printf("i=%d nf5=%lf\n",i,nf5[i]);
        }
      if(nf6[i] < 0.00)
        {
          printf("i=%d nf6=%lf\n",i,nf6[i]);
        }

      floc[i] = nf0[i]+nf1[i]+nf2[i]+nf3[i]+nf4[i]+nf5[i]+nf6[i];
    }
  gradient(floc,fgradx,fgrady);
  laplacian(floc,flap);
  if(key_interaction == 3)
    {
      for(i=1; i<=nnodes; i++)
        {
          switch(boundary_mode[i])
```



```c
            {
            case 0:
              edip[i] = floc[nv1[i]] *
                (mm-3.0*(mx*ecx[1]+my*ecy[1])*(mx*ecx[1]+my*ecy[1])) +
              floc[nv2[i]] *
                (mm-3.0*(mx*ecx[2]+my*ecy[2])*(mx*ecx[2]+my*ecy[2])) +
              floc[nv3[i]] *
                (mm-3.0*(mx*ecx[3]+my*ecy[3])*(mx*ecx[3]+my*ecy[3])) +
              floc[nv4[i]] *
                (mm-3.0*(mx*ecx[4]+my*ecy[4])*(mx*ecx[4]+my*ecy[4])) +
              floc[nv5[i]] *
                (mm-3.0*(mx*ecx[5]+my*ecy[5])*(mx*ecx[5]+my*ecy[5])) +
              floc[nv6[i]] *
                (mm-3.0*(mx*ecx[6]+my*ecy[6])*(mx*ecx[6]+my*ecy[6]));
              break;
            case 1:
              edip[i] = floc[nv1[i]] *
                (mm-3.0*(mx*ecx[1]+my*ecy[1])*(mx*ecx[1]+my*ecy[1])) +
              floc[nv4[i]] *
                (mm-3.0*(mx*ecx[4]+my*ecy[4])*(mx*ecx[4]+my*ecy[4])) +
              floc[nv5[i]] *
                (mm-3.0*(mx*ecx[5]+my*ecy[5])*(mx*ecx[5]+my*ecy[5])) +
              floc[nv6[i]] *
                (mm-3.0*(mx*ecx[6]+my*ecy[6])*(mx*ecx[6]+my*ecy[6]));
              break;
            case 2:
              edip[i] = floc[nv1[i]] *
                (mm-3.0*(mx*ecx[1]+my*ecy[1])*(mx*ecx[1]+my*ecy[1])) +
              floc[nv2[i]] *
                (mm-3.0*(mx*ecx[2]+my*ecy[2])*(mx*ecx[2]+my*ecy[2])) +
              floc[nv3[i]] *
                (mm-3.0*(mx*ecx[3]+my*ecy[3])*(mx*ecx[3]+my*ecy[3])) +
              floc[nv4[i]] *
                (mm-3.0*(mx*ecx[4]+my*ecy[4])*(mx*ecx[4]+my*ecy[4]));
              break;
            }
          edip[i] *= floc[i];
        }
      gradient(edip,egradx,egrady);
    }
  for(i=1; i<=nnodes; i++)
    {
      uxloc = (nf1[i]*ecx[1]+nf2[i]*ecx[2]+nf3[i]*ecx[3]+
               nf4[i]*ecx[4]+nf5[i]*ecx[5]+nf6[i]*ecx[6])/floc[i];
      uyloc = (nf1[i]*ecy[1]+nf2[i]*ecy[2]+nf3[i]*ecy[3]+
               nf4[i]*ecy[4]+nf5[i]*ecy[5]+nf6[i]*ecy[6])/floc[i];

      uloc2 = uxloc*uxloc + uyloc*uyloc;

      ca = (floc[i]*temp) / (1.0 - floc[i]*bpsi) - apsi * floc[i] * floc[i] -
        kappa * floc[i] * flap[i];

      cazero = floc[i] - 2.00 * ca;
      ca /= 3.00;
      cb = floc[i]/3.;
      cc = -floc[i]/6.;
      cczero = -floc[i];
      cd = 2.*floc[i]/3.;
      cgxx = kappa*(fgradx[i]*fgradx[i]-fgrady[i]*fgrady[i])/3.0;
      cgyy = -cgxx;
      cgxy = 4.00*kappa*fgradx[i]*fgrady[i]/3.;

      uxxloc = uxloc*uxloc;
      uxyloc = uxloc*uyloc;
      uyyloc = uyloc*uyloc;

      feq[0] = cazero + cczero * uloc2;

      if(feq[0] < 0.00)
```

```
        {
          printf("iter=%d i=%d cazero=%lf ca=%lf 6ca=%lf cczero=%lf uloc2=%lf feq=%lf\n"
,
                 iter,i,cazero,ca,6.0*ca,cczero,uloc2,feq[0]);
        }

      switch(key_interaction)
        {
        case 0:
          for(j=1;j<=6;j++)
            {
              feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) + cc*uloc2 +
                cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+uyyloc*ecyy[j]) +
                  cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j];
            }
          break;
        case 1: /* acceleration */
/*

          for(j=1;j<=6;j++)
            {
              feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) + cc*uloc2 +
                cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+uyyloc*ecyy[j]) +
                  fx*ecx[j] + fy*ecy[j] +
                    cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j];
            }
*/
          break;
        case 2: /* wall interaction */
          fx = tau3*floc[i]*ax;
          fy = tau3*floc[i]*ay;
          switch(boundary_mode[i])
                  {
                  case 0: /* bulk */
                    for(j=1;j<=6;j++)
                      {
                        feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) +
                          cc*uloc2 +
                          fx*ecx[j] + fy*ecy[j] +
                            cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+uyyloc*ecyy[j])
                              + cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j];
                      }
                    break;
                  case 1: /* wall 1 */
                    for(j=1;j<=6;j++)
                      {
                        if(floc[i] > 3.5) /* liquid */
                          {
                            feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) +
                              cc*uloc2 +
                              fx*ecx[j] + fy*ecy[j] +
                                cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+
                                    uyyloc*ecyy[j])
                                  + cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j] -
                                    tau3 * floc[i] * gradmiu1wall1 * ecy[j];
                          }
                        else /* vapor */
                          {
                            feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) +
                              cc*uloc2 +
                              fx*ecx[j] + fy*ecy[j] +
                                cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+
                                    uyyloc*ecyy[j])
                                  + cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j] -
                                    tau3 * floc[i] * gradmiu2wall1 * ecy[j];
                          }
                      }
                    break;
                  case 2: /* wall 2 */
```



```
                    for(j=1;j<=6;j++)
                      {
                        if(floc[i] > 3.5) /* liquid */
                          {
                            feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) +
                              cc*uloc2 +
                              fx*ecx[j] + fy*ecy[j] +
                                cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+
                                    uyyloc*ecyy[j])
                                  + cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j] -
                                    tau3 * floc[i] * gradmiu1wall2 * ecy[j];
                          }
                        else /* vapor */
                          {
                            feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) +
                              cc*uloc2 +
                              fx*ecx[j] + fy*ecy[j] +
                                cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+
                                    uyyloc*ecyy[j])
                                  + cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j] -
                                    tau3 * floc[i] * gradmiu2wall2 * ecy[j];
                          }
                      }
                    break;
                  }
          break;
        case 3: /* dipolar interaction */
          switch(boundary_mode[i])
            {
            case 0:
              fx = - tau3 * floc[i] * egradx[i];
              fy = - tau3 * floc[i] * egrady[i];
              break;
            case 1:
              if(floc[i] > 3.5) /* liquid */
                {
                  fx = tau3 * floc[i] * egradx[i];
                  fy = tau3 * floc[i] * (gradmiu1wall1 * ecy[i] + egrady[i]);
                }
              else
                {
                  fx = tau3 * floc[i] * egradx[i];
                  fy = tau3 * floc[i] * (gradmiu2wall1 * ecy[i] + egrady[i]);
                }
              break;
            case 2:
              if(floc[i] > 3.5) /* liquid */
                {
                  fx = tau3 * floc[i] * egradx[i];
                  fy = tau3 * floc[i] * (gradmiu1wall2 * ecy[i] + egrady[i]);
                }
              else
                {
                  fx = tau3 * floc[i] * egradx[i];
                  fy = tau3 * floc[i] * (gradmiu2wall2 * ecy[i] + egrady[i]);
                }
              break;
            }
          for(j=1;j<=6;j++)
            {
              feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) + cc*uloc2 +
                cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+uyyloc*ecyy[j]) +
                  cgxx*(ecxx[j]-ecyy[j])+cgxy*ecxy[j] -
                    fx * ecx[j] - fy * ecy[j];
            }
          break;
        }

      nff0[i] = nf0[i] - (nf0[i]-feq[0])/tau;
```

```
        switch(boundary_mode[i])
          {
          case 0:
            nff1[nv1[i]] = nf1[i] - (nf1[i]-feq[1])/tau;
            nff4[nv4[i]] = nf4[i] - (nf4[i]-feq[4])/tau;
            nff2[nv2[i]] = nf2[i] - (nf2[i]-feq[2])/tau;
            nff3[nv3[i]] = nf3[i] - (nf3[i]-feq[3])/tau;
            nff5[nv5[i]] = nf5[i] - (nf5[i]-feq[5])/tau;
            nff6[nv6[i]] = nf6[i] - (nf6[i]-feq[6])/tau;
            break;
          case 1:
            nff1[nv1[i]] = nf1[i] - (nf1[i]-feq[1])/tau;
            nff4[nv4[i]] = nf4[i] - (nf4[i]-feq[4])/tau;
            nff5[i] = nf2[i] - (nf2[i]-feq[2])/tau;
            nff6[i] = nf3[i] - (nf3[i]-feq[3])/tau;
            nff5[nv5[i]] = nf5[i] - (nf5[i]-feq[5])/tau;
            nff6[nv6[i]] = nf6[i] - (nf6[i]-feq[6])/tau;
            break;
          case 2:
            nff1[nv1[i]] = nf1[i] - (nf1[i]-feq[1])/tau;
            nff4[nv4[i]] = nf4[i] - (nf4[i]-feq[4])/tau;
            nff2[nv2[i]] = nf2[i] - (nf2[i]-feq[2])/tau;
            nff3[nv3[i]] = nf3[i] - (nf3[i]-feq[3])/tau;
            nff2[i] = nf5[i] - (nf5[i]-feq[5])/tau;
            nff3[i] = nf6[i] - (nf6[i]-feq[6])/tau;
            break;
/*
          case 3:
            ff1[nv1[i]] = f1[i] - (f1[i]-feq[1])/tau;
            ff2[nv2[i]] = f2[i] - (f2[i]-feq[2])/tau;
            ff6[nv6[i]] = f6[i] - (f6[i]-feq[6])/tau;
            ff1[i] = f4[i] - (f4[i]-feq[4])/tau;
            ff2[i] = f5[i] - (f5[i]-feq[5])/tau;
            ff6[i] = f3[i] - (f3[i]-feq[3])/tau;
            break;
          case 4:
            ff4[nv4[i]] = f4[i] - (f4[i]-feq[4])/tau;
            ff2[nv2[i]] = f2[i] - (f2[i]-feq[2])/tau;
            ff3[nv3[i]] = f3[i] - (f3[i]-feq[3])/tau;
            ff5[nv5[i]] = f5[i] - (f5[i]-feq[5])/tau;
            ff6[nv6[i]] = f6[i] - (f6[i]-feq[6])/tau;
            ff4[i] = f1[i] - (f1[i]-feq[1])/tau;
            if((ff0[i]<0.00)||(ff1[i]<0.00)||(ff2[i]<0.00)||(ff3[i]<0.00)||
               (ff4[i]<0.00)||(ff5[i]<0.00)||(ff6[i]<0.00))
              {
                frez = fopen(rez_name,"a");
                fprintf(frez,"iter=%d i=%d\n",iter,i);
                fprintf(frez,"ff < 0. ===>  hiba\n");

                fprintf(frez,"wmiux[i]=%20.16f wmiuy[i]=%20.16f\n
                nfx=%20.16f nfy=%20.16f\nfx=%20.16f fy=%20.16f\n",
                wmiux[i],wmiuy[i],nloc[i]*force_x,nloc[i]*force_y,fx,fy);

                fprintf(frez,"ecx=%f %f %f %f %f %f\n",ecx[1],ecx[2],ecx[3],
                        ecx[4],ecx[5],ecx[6]);
                fprintf(frez,"ecy=%f %f %f %f %f %f\n",ecy[1],ecy[2],ecy[3],
                        ecy[4],ecy[5],ecy[6]);
                fprintf(frez,"f0[i]=%f ff0[i]=%f feq[0]=%f\n",f0[i],ff0[i],feq[0]);
                fprintf(frez,"f1[i]=%f ff1[i]=%f feq[1]=%f\n",f1[i],ff1[i],feq[1]);
                fprintf(frez,"f2[i]=%f ff2[i]=%f feq[2]=%f\n",f2[i],ff2[i],feq[2]);
                fprintf(frez,"f3[i]=%f ff3[i]=%f feq[3]=%f\n",f3[i],ff3[i],feq[3]);
                fprintf(frez,"f4[i]=%f ff4[i]=%f feq[4]=%f\n",f4[i],ff4[i],feq[4]);
                fprintf(frez,"f5[i]=%f ff5[i]=%f feq[5]=%f\n",f5[i],ff5[i],feq[5]);
                fprintf(frez,"f6[i]=%f ff6[i]=%f feq[6]=%f\n",f6[i],ff6[i],feq[6]);
                fclose(frez);
              }
            break;
          case 5:
```



```
            ff1[nv1[i]] = f1[i] - (f1[i]-feq[1])/tau;
            ff2[nv2[i]] = f2[i] - (f2[i]-feq[2])/tau;
            ff3[nv3[i]] = f3[i] - (f3[i]-feq[3])/tau;
            ff5[nv5[i]] = f5[i] - (f5[i]-feq[5])/tau;
            ff6[nv6[i]] = f6[i] - (f6[i]-feq[6])/tau;
            ff1[i] = f4[i] - (f4[i]-feq[4])/tau;
            break;
          case 6:
            ff3[nv3[i]] = f3[i] - (f3[i]-feq[3])/tau;
            ff4[nv4[i]] = f4[i] - (f4[i]-feq[4])/tau;
            ff5[nv5[i]] = f5[i] - (f5[i]-feq[5])/tau;
            ff3[i] = f6[i] - (f6[i]-feq[6])/tau;
            ff4[i] = f1[i] - (f1[i]-feq[1])/tau;
            ff5[i] = f2[i] - (f2[i]-feq[2])/tau;
            break;
          case 7:
            ff1[nv1[i]] = f1[i] - (f1[i]-feq[1])/tau;
            ff5[nv5[i]] = f5[i] - (f5[i]-feq[5])/tau;
            ff6[nv6[i]] = f6[i] - (f6[i]-feq[6])/tau;
            ff1[i] = f4[i] - (f4[i]-feq[4])/tau;
            ff5[i] = f2[i] - (f2[i]-feq[2])/tau;
            ff6[i] = f3[i] - (f3[i]-feq[3])/tau;
            break;
          case 8:
            ff4[nv4[i]] = f4[i] - (f4[i]-feq[4])/tau;
            ff5[nv5[i]] = f5[i] - (f5[i]-feq[5])/tau;
            ff3[i] = f6[i] - (f6[i]-feq[6])/tau;
            ff4[i] = f1[i] - (f1[i]-feq[1])/tau;
            ff5[i] = f2[i] - (f2[i]-feq[2])/tau;
            ff6[i] = f3[i] - (f3[i]-feq[3])/tau;
            break;
          case 9:
            ff1[nv1[i]] = f1[i] - (f1[i]-feq[1])/tau;
            ff2[nv2[i]] = f2[i] - (f2[i]-feq[2])/tau;
            ff1[i] = f4[i] - (f4[i]-feq[4])/tau;
            ff2[i] = f5[i] - (f5[i]-feq[5])/tau;
            ff3[i] = f6[i] - (f6[i]-feq[6])/tau;
            ff6[i] = f3[i] - (f3[i]-feq[3])/tau;
            break;
          case 10:
            ff2[nv2[i]] = f2[i] - (f2[i]-feq[2])/tau;
            ff3[nv3[i]] = f3[i] - (f3[i]-feq[3])/tau;
            ff4[nv4[i]] = f4[i] - (f4[i]-feq[4])/tau;
            ff2[i] = f5[i] - (f5[i]-feq[5])/tau;
            ff3[i] = f6[i] - (f6[i]-feq[6])/tau;
            ff4[i] = f1[i] - (f1[i]-feq[1])/tau;
            break;
*/
          }
      }
}


void automaton(void)
{
  int autiter;
  autiter = 0;
  while( autiter < niter_cycle )
    {
      automaton_aux(f0, f1, f2, f3, f4, f5, f6,
                    ff0, ff1, ff2, ff3, ff4, ff5, ff6);
      automaton_aux(ff0, ff1, ff2, ff3, ff4, ff5, ff6,
                    f0, f1, f2, f3, f4, f5, f6);
      autiter++;
      iter++;
    }
}
```

```
#define MAIN_HEADER

#include <stdio.h>
#include <string.h>

#include "swhead.h"

void sw_input(void);
void sw_output(void);
void build_names(int);
void getavec_hex(void);
void xv(double n0[], double n1[], double n2[], double n3[],
        double n4[], double n5[], double n6[], char filename[]);
void init_arrays_aux(void);
void init_arrays(void);
void automaton(void);

void main(void)
{
  int isim, isav, icycle;
  FILE *fin, *fout, *fsav;

  sw_input();
  sw_output();

  for(isim=0; isim<nsim; isim++)
    {
      switch(akey_status[isim])
        {
        case 0:
          build_names(isim);
          getavec_hex();
          key_init = akey_init[isim];
          key_interaction = akey_interaction_ini[isim];
          key_boundary = akey_boundary_ini[isim];
          temp_in = atemp_in[isim];
          temp_fin = atemp_fin[isim];
          kappa = akappa[isim];
          tau = atau[isim];
          tau3 = tau / 3.00;
          ax = aax[isim];
          ay = aay[isim];
          mx = amx[isim];
          my = amy[isim];
          mm = mx*mx + my*my;
          gradmiu1wall1 = agradmiu1wall1[isim];
          gradmiu2wall1 = agradmiu2wall1[isim];
          gradmiu1wall2 = agradmiu1wall2[isim];
          gradmiu2wall2 = agradmiu2wall2[isim];
          init_arrays_hex();
          init_arrays_hex_aux();
          niter_init = aniter_init[isim];
          niter_cycle = niter_init;
          temp = temp_in;
          iter = 0;
          xv(f0,f1,f2,f3,f4,f5,f6,xv_name);
/*        profile(f0,f1,f2,f3,f4,f5,f6,profile_name);  */
          automaton();
          iter = 0;
          xv(f0,f1,f2,f3,f4,f5,f6,xv_name);
          minko();
/*        profile(f0,f1,f2,f3,f4,f5,f6,profile_name);  */
          key_interaction = akey_interaction_fin[isim];
          key_boundary = akey_boundary_fin[isim];
          init_arrays_hex_aux();
          nsav = ansav[isim];
          ncycles = ancycles[isim];
          niter_cycle = aniter_cycle[isim];
          temp = temp_fin;
```



```
          for(isav=0; isav<nsav; isav++)
            {
              for(icycle=0; icycle<ncycles; icycle++)
                {
/*                printf("isav=%d icycle=%d\n",isav,icycle);   */
                  automaton();
                  xv(f0,f1,f2,f3,f4,f5,f6,xv_name);
                  minko();
/*                profile(f0,f1,f2,f3,f4,f5,f6,profile_name);  */
                }
            }
        }
    }
}
```

```
1
0 Drop-mx
10 1 100 1000
7 0 2 0 1
3.5 0.550 0.550 0.0094 0.8
0.00 0.0000000 0.00 0.00
0.00 0.00 0.01 0.000
```